UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY
A INFORMATIKY

BAKALÁŘSKÁ PRÁCE

2008 Martin Šmejda

UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Redundance dat na datových nosičích pomocí Reed-Solomonových kódů

Martin Šmejda

Bakalářská práce
2008

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Katedra informačních technologií
Akademický rok: **2007/2008**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Martin ŠMEJDA**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Redundance dat na datových nosičích pomocí Reed-Solomonových kódů**

Zásady pro vypracování:

Teoretická část:
- Porovnání základních používaných algoritmů a technik zajišťující redundanci dat
- Teoretický rozbor funkce Reed-Solomonových kódů (algoritmů)
- Složitost a výpočetní náročnost algoritmů
- Rozbor programu a v něm použitých algoritmů a datových struktur

Implementační část:
- Implementace vybraných algoritmů pro vytvoření Reed-Solomonových opravných kódů
- Využití výše uvedených alogoritmů v programu umožnujícím zálohování a obnovu dat
- Demonstrace funkce programu na různých typech dat a datových nosičů

Požadavky:
- Efektivnost programu (optimalizace) kvůli velké výpočetní náročnosti R-S kódu
- Vysoká použitelnost v běžném světě
- Jednoduchost, User-Friendly, GUI rozhraní
- Použitý programovací jazykc – C++, QT

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

[1] SKLAR, Bernard. Reed-Solomon Codes [online]. 2002. Dostupný z WWW: <http://www.informit.com/content/images/art_sklar7_reed-solomon/elementLinks/art_sklar7_reed-solomon.pdf>.

[2] GNöRLICH, Carsten. Project Dvdisaster. 2006. Překlad Staněk Luboš. Dostupný z WWW: <http://dvdisaster.net/legacy/cs/index.html>.

[3] Koton, J., Číka, P., Křivánek, V. Samoopravné Reed-Solomonovy kódy. 2006. Dostupné z WWW: <http://access.feld.cvut.cz/view.php?nazevclanku=samoopravne-reed-solomonovy-kody&cisloclanku=2006080002>. ISSN 1214-9675.

[4] ČÍKA, P., KodekBCH kódu. Diplomová práce. Ústav telekomunikací - FEKT VUT v Brně, 2005.

Vedoucí bakalářské práce: **Ing. Petr Veselý**
Katedra informatiky v dopravě

Datum zadání bakalářské práce: **30. listopadu 2007**

Termín odevzdání bakalářské práce: **16. května 2008**

Univerzita Pardubice Česká republika

L.S.

doc. Ing. Simeon Karamazov, Dr.
děkan

V Pardubicích dne 29. dubna 2008

SOUHRN

Práce je primárně zaměřena na redundanci dat na datových nosičích, kdy při ztrátě nebo poškození záznamu na jednom nebo několika nosičích, tato skutečnost nebude znamenat ztrátu dat na nich uložených. V teoretické části se zabývám dnes nejběžnějším způsobem redundance dat „za běhu“ kterým jsou RAIDová pole, jejich typy a implementace. Druhá část se zabývá Reed-Solomonovými opravnými kódy, jejich teorií doplněnou názornými příklady funkce jednotlivých kroků algoritmu. Praktická část se zabývá implementací Reed-Solomonova opravného kódu v programu, který aplikuje tyto opravné kódy na jakýkoli typ dat.

KLÍČOVÁ SLOVA

zálohování dat, Reed-Solomonovy opravné kódy, RAID, QT, c++

TITLE

Data redundancy on data mediums by Reed-Solomon codes

ABSTRACT

This work is primary oriented on data redundancy on data medium when by loss of record on one or several mediums it do not leed to loss of stored data itself. Theoretical part of the work focus on todays most common way of data redundancy „on the fly“ which are RAID fields, its types and implementation. Second part focus on Reed-Solomon error correction code, its theory supplemented by vivid examples of individual steps of the algoritm. Practical part focus on implementation of Reed-Solomon code into the program that apply the error correcting code on any data.

KEY WORDS

data redundancy, Reed-Solomon error correction code, RAID, QT, c++

Obsah

1 Úvod

Tuto práci jsem vypracoval z důvodu mojí záliby v RAID polích a možnosti vytvořit program, který díky implementaci Reed-Solomonových kódů dokáže opravit poškozené soubory například z částečně poškozeného CD nosiče. Chtěl jsem také využít svoje čerstvě nabyté znalosti z předmětu c++ 3 a to práci s QT knihovnami, které můj program používá, jako grafické rozhraní a pro práci se soubory.

2 Redundance dat pomocí techniky RAID

Tyto informace včetně obrázků jsou ze zdrojů wikipedie [3] a [4].

2.1 Co je to RAID

RAID – Redundant Array of Independent Disks (vícenásobné diskové pole nezávislých disků) je typ diskových řadičů, které zabezpečují pomocí určitých speciálních funkcí koordinovanou práci dvou nebo více fyzických diskových jednotek. Zvyšuje se tak výkon a odolnost vůči chybám nebo ztrátě dat.

Existuje celkem šest typů polí, prakticky se používají tři typy.

Disková pole typu RAID jsou velmi často používány na serverech. U osobních počítačů se teď stávají čím dál oblíbenější (obzvláště typy 0 a 1), hlavně kvůli ceně disků a jejich mnohdy nevalné spolehlivosti.

Pojem RAID vznikl v roce 1988, kdy byla Univerzitou California – Berkeley vydána publikace A Case For Redundant Arrays of Inexpensive Disks (David Patterson, Randy Katz a Garth Gibson) [26]. 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 sériových pevných disků, které nejsou nijak upravovány.

2.2 Typy RAID

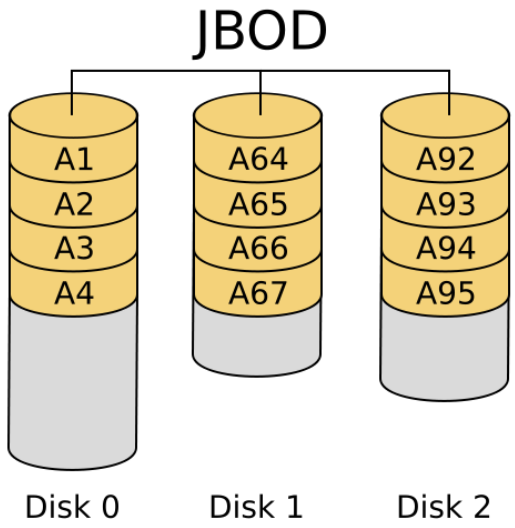
Používá se několik úrovní RAID, rozdělené podle vyšší redundance dat nebo vyššího výkonu nebo kombinací obojího.

2.2.1 RAID 0 (minimálně 2 disky)

2 typy: zřetězení, prokládání (striping)

zřetězení: (někdy označovaný jako Lineární mód) data jsou postupně ukládána na několik disků. Jakmile se zaplní první, ukládá se na druhý, poté na třetí atd. Výhodou je, že získáme velký logický disk, nevýhodou, že data nejsou po pádu jednoho disku obnovitelná. Bývá též označováno jako JBOD - Just a Bunch Of Disks (jen svazek

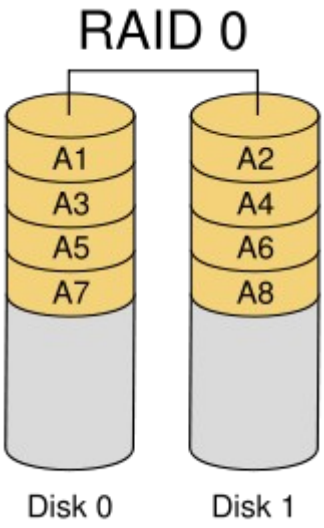
disků), „D“ bývá někdy interpretováno jako Drives (jen svazek jednotek). Není to skutečný RAID level, ovšem byla by chyba toto zapojení zde nezmínit. Následující ilustrace 1 popisuje funkci. Data se začínají ukládat na první disk A1 až A63 (není vyobrazen), když se tento disk zaplní pokračuje se na dalším blokem A64 atd. Disky jsou nestejné velikosti.



Ilustrace 1: JBOD

prokládání: data jsou ukládána na disky prokládaně. To znamená, že soubor je rozdělen na menší části (bloky), jejichž velikost je volitelná (vždy mocnina dvou) a každá část je ukládána střídavě na všechny disky. Diskové pole se tak opět jeví jako jeden velký disk. Není odolný vůči chybám. Porucha jednoho disku znamená ztrátu všech dat v diskovém poli, protože jeden soubor je na více fyzických discích. Výrazně se ale zvyšuje rychlost čtení, viz kapitola 1.2.4 a Ilustrace 2, protože čteme zároveň z několika disků najednou. Zato zápis může být pomalejší, protože se ukládají stejná data na dva disky, záleží na konkrétním systému, připojení disků k řadičům apod.

Příklad: Máme soubor A o velikosti 32 kB, který ukládáme na diskové pole v RAID levelu 0, velikost bloku dat je 4 kB, v poli jsou přítomny dva disky. Soubor je rozdělen na 32/4 = 8 částí a ty jsou pravidelným střídáním rozděleny na disky, tedy první část A1 na první disk, druhá část A2 na druhý disk, třetí část A3 znovu na první disk atd. Viz ilustrace 2.

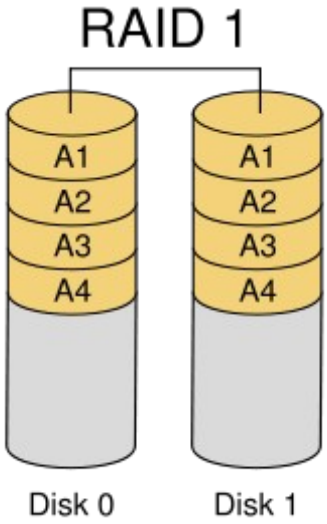


Ilustrace 2: Raid 0

2.2.2 RAID 1 – zrcadlení (minimálně 2 disky)

Nejjednodušší ale poměrně efektivní ochrana dat. Provádí se zrcadlení (mirroring) obsahu disků. Pro zapojení jsou třeba dva až N disků kde N je násobek dvou, velmi obvykle právě dva. Obsah se současné zaznamenává na dva disky. V případě výpadku jednoho disku se pracuje s kopií, která je ihned k dispozici. Podobná technika může být uplatněna o úroveň výše, kdy jsou použity dva samostatné řadiče. Tato technika se nazývá duplexing a je odolná i proti výpadku řadiče. Obě techniky buď nemění nebo snižují rychlost, záleží na zapojení disků k řadiči. Např. PATA disky zapojené k jednomu kanálu se musí o sběrnici dělit, proto při zápisu (zapisují se vždy stejná data na oba) může docházet ke snížení rychlosti proti zapojení s jedním diskem, konkrétní příklad viz kapitola 1.2.4. Na systému optimalizovaném pro RAID 1 může dojít k zrychlení čtení (Linux) kdy se části jednoho souboru čtou ne jen z jednoho, ale z více disků naráz. RAID 1 výrazně zvyšuje bezpečnost dat proti ztrátě způsobené poruchou hardware. Pokud bereme v úvahu disky s týdenní pravděpodobností selhání 1:500 RAID 1 pravděpodobnost selhání takového pole zvýší na $(1/500)^2 = 1/250\ 000$, tedy pětisetnásobně. Nevýhodou je potřeba dvojnásobné diskové kapacity.

Příklad: Máme soubor A o velikosti 32 kB, který ukládáme na diskové pole v RAID levelu 0, velikost clusteru[1] je 8 kB, v poli jsou přítomny dva disky. Soubor je rozdělen na 32/8 = 4 clustery a ty jsou jeden za druhým zapisovány na disky, tedy první část A1 a první disk, kopie první části na druhý disk, druhá část A2 na první disk, kopie druhé části na druhý disk atd. Viz ilustrace 3.



Ilustrace 3: RAID 1

2.2.3 Kombinace RAID 0 a 1

Pole RAID 0+1 prokládaný (striping) set v zrcadleném (mirroring) setu je kombinací RAID 0 a RAID 1. Data uložíme prokládaně na dva disky (A, B), poté totéž uděláme s dalšími dvěma disky(C, D). Získáme tak dva logické disky AB, CD, které mají redundantní obsah. (Máme-li soubor, který se při prokládání rozdělí na dvě poloviny, první část souboru máme na disku A a C, druhou na disku B a D) Výhodou tohoto způsobu je, že nejen rozkládáme zátěž mezi více disků při čtení a zápisu, ale data jsou uložena redundantně, takže se dají po chybě snadno obnovit. Mezi nevýhody patří využití pouze 50 % celkové diskové kapacity, a při výpadku jednoho nebo více disků ze stejného setu (z jedné strany zrcadlení) ztrácíme redundantnost dat a při výpadku dvou a více disků z obou stran zrcadlení ztrácíme veškerá data.

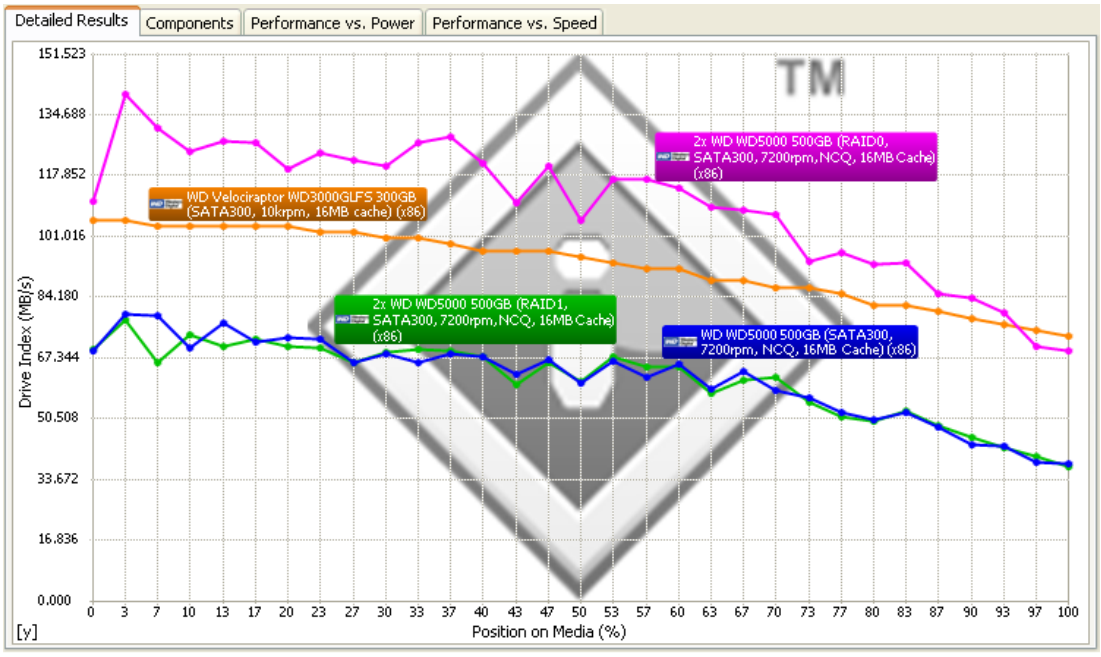
Pole RAID 1+0 zrcadlený set v prokládaném setu je opět kombinací RAID 0 a RAID 1, ale postupujeme obráceně než v předchozím případě. Nejdříve uložíme

1 cluster: nejmenší blok dat, který lze na disk zapsat, sestává se z jednoho nebo více (vždy mocnina dvou) sektorů, sektor je 512 B dat.

stejná data na disk A, B, poté na disk C, D. Získáme tak dva logické disky AB, CD, na nichž jsou data uložena prokládaně. (Máme-li soubor, který se při prokládání rozdělí na dvě poloviny, první část souboru je na disku A a B, druhá část je na disku C a D, na rozdíl od RAID 0-1. Výhody jsou podobné RAID 0-1, navíc je RAID 1-0 odolnější proti výpadku více disků a po chybě je obnova dat mnohem rychlejší. Nevýhodou je opět využití pouze 50 % kapacity.

2.2.4 RAID 0 a 1 a rychlost čtení

Následující graf je pořízen z programu pro měření výkonu PC komponent SiSoftware Sandra Lite 2008.5.14.24, konkrétně jde o výkon při čtení z fyzických disků. Modrá křivka reprezentuje 500 GB disk zapojený samostatně bez členství v RAID poli. Zelená křivka znázorňuje průběh rychlosti čtení u disku stejné kapacity, ale zapojeného ještě s jedním diskem stejného typu do RAID pole 1. Fialový graf znázorňuje stejné dva disky jako v předchozím případě jen zapojené do pole typu RAID 0. Pro názornost je zde ještě oranžovou křivkou znázorněn průběh čtení dnes nejrychlejšího (květen 2008) desktopového disku Western Digital Velociraptor WD3000GLFS o kapacitě 300 GB a otáčkách ploten 10 000 ot/s. U RAID 0 zapojení je zřejmý drastický nárůst rychlosti čtení, téměř dvojnásobný proti neRAID zapojení. Oproti tomu na platformě Windows XP u RAID 1 není vidět téměř žádný rozdíl v rychlosti čtení proti zapojení jen jednoho disku, viz ilustrace 4.



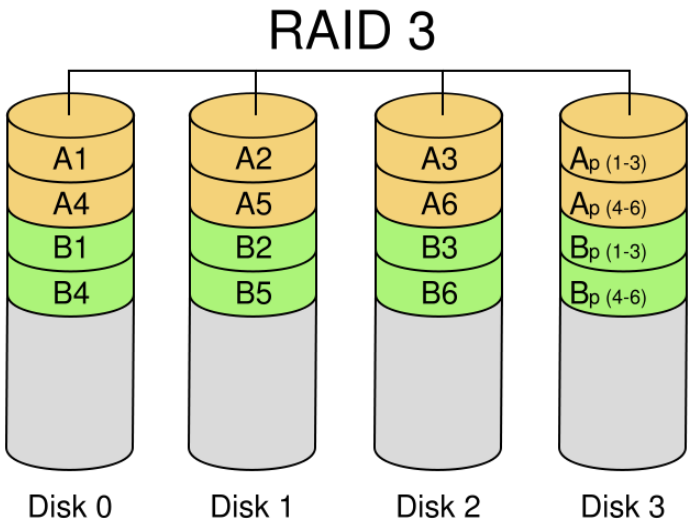
Ilustrace 4: SiSoftware Sandra test rychlosti čtení fyzických disků

- Zelená křivka: 1x WD5000AAKS, 500 GB, 7200 ot/s.
- Modrá křivka: 2x WD5000AAKS, 500 GB, 7200 ot/s, RAID 1.
- Fialová křivka: 2x WD5000AAKS, 500 GB, 7200 ot/s, RAID 0.
- Oranžová křivka: 1x WD3000-GLFS, 300 GB, 10 000 ot/s.

2.2.5 RAID 3 (minimálně 3 disky)

Je použito N+1 stejných disků. Na N disků jsou ukládána data a na poslední disk je uložen exkluzivní součin (XOR) neboli parita těchto dat. Při výpadku paritního disku jsou data zachována, při výpadku libovolného jiného disku je možno z ostatních disků spolu s paritním diskem ztracená data zrekonstruovat. Výhodou je potřeba jen jednoho disku navíc. Navíc se zkracuje doba odpovědi stejně jako u RAID 2 (RAID 2 je jediný původní RAID level, který se v současnosti nepoužívá, více [24]). Nevýhoda je, že paritní disk je takzvaný bottle neck (hrdlo láhve), neboli problémové místo systému a je vytížen při zápisu na jakýkoliv jiný disk. RAID 3 se používá velmi málo, protože tím, že data se mezi disky dělí po jednom bytu je většinou jeden soubor rozprostřen po všech discích pole a není možné k souborům přistupovat souběžně.

Příklad: Bereme v úvahu dva soubory A a B, oba mají stejnou délku 6 B, velikost bloku dat je jeden Byte, v poli jsou přítomny tři disky + jeden paritní. Soubor se tedy rozdělí na 6/1 = 6 částí, kde se část A1 uloží na první disk, část A2 na druhý, část A3 na třetí disk, z těchto tří částí se vypočítá parita a ta se uloží na čtvrtý disk. To samé se provedu se zbytkem souboru A4-A6. Viz Ilustrace 5

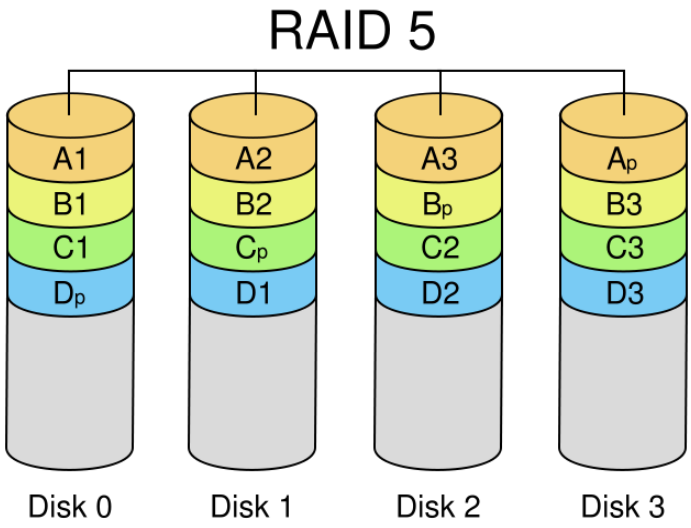


Ilustrace 5: RAID 3

2.2.6 RAID 5 (minimálně 3 disky)

Velmi podobné RAID 3, ale odstraňuje problém s přetíženým paritním diskem, neboť jsou paritní data uložena střídavě na všech discích a ne pouze na jednom. Výhodou je, že jen jeden disk (i když pokaždé jiný) obsahuje redundantní informace a opět se dá využít paralelního přístupu k diskům, čímž se zkrátí doba odpovědi. Nevýhodou RAID 5 je pomalejší zápis, kvůli nutnosti počítat paritu. Abychom mohli paritu vypočítat potřebujeme znát hodnotu všech bytů ve stejné řadě jako tam kde právě zapisujeme, zápis tedy vyžaduje i načtení „starých“ dat ze stejné řady ze všech disků v poli. RAID 5 se dá chápat jako speciální případ Reed-Solomonova kódu. RAID 5 jest jeho ořezaná verze vyžadující pouze součty nad Galoisovým polem (viz kapitola 2.3). Protože operujeme na bitech, pole používá binární Galoisovo pole stupně 2^1, neboli GF(2). V cyklické reprezentaci binárního Galoisova pole je sčítání počítané jako obyčejný exkluzivní součet (XOR).

Příklad: Bereme v úvahu soubory A až D, všechny mají stejnou délku 12 kB, velikost bloku dat je 4 kB, v poli jsou přítomny čtyři disky. Soubor se tedy rozdělí na 12/4 = 3 části, kde se část A1 uloží na první disk, část A2 na druhý, část A3 na třetí disk, z těchto tří částí se vypočítá parita a ta se uloží na čtvrtý disk. To samé se provedu se B, jen parita se distribuuje na jiný disk než v předchozím případě, tentokráte na disk třetí. Viz Ilustrace 6

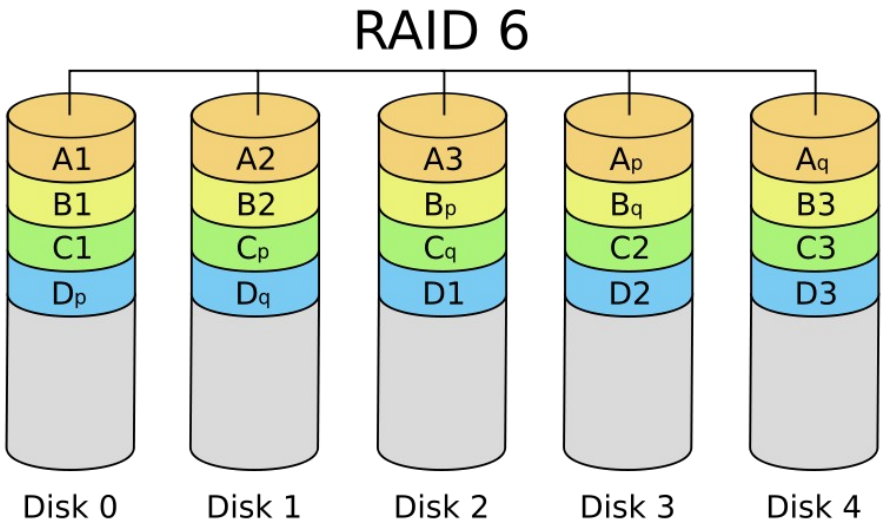


Ilustrace 6: RAID 5

2.2.7 RAID 6 (minimálně 4 disky)

Podobné jako RAID 5 jen používá duální paritu, tím poskytuje chybovou toreranci při selhání dvou ze čtyř disků, výhodné použití hlavně u vysokokapacitních polí (více než 1 TB), kde obnovení dat na velkých discích trvá dlouho. Pole v RAID 6 je i s 3-mi disky schopno normálně pracovat zatímco se čtvrtý disk rebuilduje po selhání. Prvek Aq na Ilustraci 7 je syndrom vypočítaný z dat A1 až A3 typicky ve formě polynomu nad Galoisovým polem GF(2^8) viz kapitola 2.10.

Postup ukládání je prakticky totožný jako u RAID 5, pouze se navíc počítá datový syndrom.



Ilustrace 7: RAID 6

2.2.8 Jak funguje XOR a počítání parity

Protože se XOR týká RAIDu 3, 5 a 6 následující odstavec popisuje jak funguje. XOR je zkratka pro bitový exkluzivní součet. Funguje tím způsobem, že pokud máme dvě stejné hodnoty např. 1 a 1 nebo 0 a 0, výsledek je 0. Pokud se hodnoty liší, v binárním hledisku je na výběr 0 a 1 nebo 1 a 0, výsledek operace je 1. Následující tabulka 1 přehledně popisuje funkci.

Hodnota 1	Hodnota 2	Výsledek (XOR)
0	0	0
1	0	1
0	1	1
1	1	0

Tabulka 2.1: Exkluzivní součet

Raid 3, 4 a 5 používá k vypočítání parity operaci XOR (matematický zápis je $\oplus$), která vytvoří sudou praritu. To znamená že součet všech jedniček na stejných pozicích je vždy sudý. Máme dané byty A_1, A_2, A_3 jejichž hodnota je zobrazena níže, ty to byty postupně XORujeme až dostaneme výsledek. Tento příklad je plně reálný pro data na poli s RAID 3, kde se data dělí právě po bytech:

- $A_1 = 0000\ 0111$
- $A_2 = 0000\ 0101$
- $A3 = 0000\ 0000$

Operace XOR nad těmito byty bude vypadat takto:

$A_1 \oplus A_2 \oplus A_3 \oplus = (0000\ 0111 \oplus 0000\ 0101) \oplus 0000\ 0000 =$

$= 0000\ 0010 \oplus 0000\ 0000 = 0000\ 0010$

Počet jedniček na pozici nulté, první i druhé je dvě, tedy sudé číslo. Berme nyní v úvahu, že z důvodu selhání jednoho disku v poli jsme přišli o datový blok A_2, zbyl tedy datový blok A_1 a A_3 a paritní blok A_p, ztracená data dostaneme zpět opět pomocí funkce XOR:

$A_2 = A_1 \oplus A_3 \oplus A_p$

$A_2 = (A_1 \oplus A_3) \oplus A_p$

$A_2 = (0000\ 0111 \oplus 0000\ 0000) \oplus 0000\ 0010$

$A_2 = 0000\ 0111 \oplus 0000\ 0010$

$A_2 = 0000\ 0101$

Jak je vidět původní hodnotu A2 jsme korektně dostali zpátky z hodnot které zůstali neporušeny. Tento příklad naznačuje použití jednoho paritního disku a tří datových. Toto schéma však funguje stejně na jakémkoli počtu datových disků s jedním diskem paritním, ovšem při běhu mnoha datových disků se zvyšuje pravděpodobnost selhání dvou a více z nich, což je selhání už nechráněné výpočtem přes exkluzivní součet. Také většina disků operuje ne s byty, ale s blokem dat, ovšem základní bitové schéma exkluzivního součtu funguje stejně na datech o jakékoli délce slova.

Kompletní popis všech standardních RAID levelů je v literatuře [6] a nestandardních v literatuře [7].

2.3 Software RAID

Softwarové implementace jsou podporovány většinou operačních systémů. Softwarová vrstva leží nad ovladači disku a poskytuje abstraktní vrstvu mezi logickými

disky (RAID poli) a fyzickými disky. Softwarový RAID je typicky limitován na RAID 0 (stripování) a RAID 1 (zrcadlení). V multivláknových operačních systémech (jako Linux, MacOS X, Windows NT/00/XP/Vista, Nowell NetWare) systém dokáže překrývat (overlap) I/O, což umožňuje vytvářet mnohonásobné požadavky na čtení nebo zápis bez toho, aby se muselo čekat na dokončení každého požadavku. Tato schopnost umožňuje použití v RAID 0/1 v OS. Avšak většina OS nepodporuje použití RAID 0/1 stripping nebo mirroring s užitím parity kvůli druhotným požadavkům na výkon systému při počítání parity. To je hlavní výhoda a někdy nutnost použití Hardware RAIDu. Software imlementace vyžaduje jen velmi málo procesního času, poskytovaného CPU v hostujícím systému. Současné počítače poskytují většinou dostatek procesního výkonu, od doby kdy SCSI, IDE a SATA disky podporují asynchronní zápis/čtení jakýkoli vícevláknový OS může podporovat ne-paritní RAID na několika discích s velmi nízkým vzrůstem využití procesoru. Software implementace neposkytuje takovou výkonovou úroveň jako hardware založený RAID, protože software musí běžet na hostujícím serveru připojeném na úložný prostor. Implementace na software úrovni může být výhodné z hlediska ceny řešení, ovšem to záleží na konkrétním použití. Ovšem pokud server selže (kvůli selhání disku), připojený úložný prostor není po nějakou dobu přístupný, v případě Hardware RAIDu by bylo možné úložiště připojit na jiný hostující systém a náprava by byla velmi rychlá.

2.3.1 Linux Software RAID: mdadm a raidtools

Mdadm jsou s raidtools standardními nástroji na vytváření, správu a monitorování RAID polí v Linuxu. Stáhnout lze z oficiálních stránek tvůrce mdadm Neila Browna: http://www.cse.unsw.edu.au/~neilb/source/mdadm/ nebo z kernel.org http://www.kernel.org/pub/linux/utils/raid/mdadm/, v současné době ve verzi 2.6.4 z 19. listopadu 2007 nebo jako balíček pro konkrétní distribuci, instalace spočívá v rozbalení zdrojových souborů a provedení *make install.* Raidtools jsou soubor utilit pro „old-style“ RAID disky, k nalezení jsou například zde: http://people.radhat.com/mingo/raidtools/.

V současné době podporuje Linux následující RAID levely: Linear, RAID 0 (striping), RAID 1 (mirroring), RAID 4, RAID 5, RAID 6, RAID 10 a ještě další dvě MULTIPATH a FAULTY které však nejsou mody sw RAIDu.

2.3.2 Příklady konfigurace

Tyto informace jsem po většinou čerpal z [1], [2] nebo manuálu programů raid tools a mdadm prostředí Linux Fedora 8.

Linear mode: pokud nemáme stejně velké disky a chceme z nich vytvořit jeden velký virtuální.

pro raidtools: konfigurační soubor se nachází v /etc/raidtab, zde máme dva disky v lineárním modu:

```
raiddev /dev/md0
        raid-level      linear
        nr-raid-disks   2
        chunk-size      32
        persistent-superblock 1
        device          /dev/sdb6
        raid-disk       0
        device          /dev/sdc5
        raid-disk       1
```

a Lineární mod pomocí mdadm:

```
mdadm --create --verbose /dev/md0 --level=linear
--raid-devices=2 /dev/sdb6 /dev/sdc5
```

RAID 0: máme dva podobně velké disky a chceme dvojnásobný výkon:

raidtools:

```
raiddev /dev/md0
        raid-level      0
        nr-raid-disks   2
        persistent-superblock 1
        chunk-size     4
        device          /dev/sdb6
        raid-disk       0
        device          /dev/sdc5
        raid-disk       1
```

to samé v mdadm:

```
mdadm --create --verbose /dev/md0 --level=0 --raid-
devices=2 /dev/sdb6 /dev/sdc5
```

RAID 1: máme dva podobně velké disky a chceme spolehlivost:

raidtools:

```
raiddev /dev/md0
        raid-level      1
        nr-raid-disks   2
        nr-spare-disks  0
        persistent-superblock 1
        device          /dev/sdb6
        raid-disk       0
        device          /dev/sdc5
        raid-disk       1
```

pokud máme spare disk pak ještě:

```
        device          /dev/sdd5
        spare-disk      0
```

mdadm:

```
mdadm --create --verbose /dev/md0 --level=1 --raid-
devices=2 /dev/sdb6 /dev/sdc5
```

RAID 5: mame N disků a chceme redundanci pro bezpečnost dat, výkon disk. pole a zároveň máme dostatek procesního výkonu na počítání parity, zde příklad pro sedmi diskové pole:

raidtools:

```
raiddev /dev/md0
        raid-level      5
        nr-raid-disks   7
        nr-spare-disks  0
        persistent-superblock 1
        parity-algorithm        left-symmetric
        chunk-size      32
```

```
device          /dev/sda3
raid-disk       0
device          /dev/sdb1
raid-disk       1
device          /dev/sdc1
raid-disk       2
device          /dev/sdd1
raid-disk       3
device          /dev/sde1
raid-disk       4
device          /dev/sdf1
raid-disk       5
device          /dev/sdg1
raid-disk       6
```

opět pokud máme spare disk potom ještě:

```
device         /dev/sdh1
spare-disk     0
```

mdadm:

```
mdadm --create --verbose /dev/md0 --level=5 --raid-devices=7 /dev/sda3 /dev/sdb1 /dev/sdc1 /dev/sdd1 /dev/sde1 /dev/sdf1 /dev/sdg1
```

pozn.: pokud použijeme nestejně velké disky bude kapacita výsledného pole: (*počet Disků-1)*nejmenšíDisk,* mínus jedna protože vždy 1/N z celkové kapacity disků je zabrána paritou, přičemž kapacita nestejně velkých disků se redukuje na kapacitu nejmenšího použitého disku v poli.

pro vytvoření pole u raidtools je ještě třeba spustit:

```
mkraid /dev/md0
```

poté se již s polem pracuje jako s normálním diskovým oddílem, např:

```
mke2fs /dev/md0 #zformátuje pole
mount /dev/md0 /mnt/raidpole #připojí pole jako disk. oddíl
```

```
mdadm --scan #prozkoumá všechny oddíly definované
v /proc/mdstat
```

2.3.3 Vysvětlivky k pojmům v konfiguračním souboru /etc/raidtab

raiddev: tímto se definuje název svazku raidového pole, formát je md0-X

raid-level: uvádí typ použitého pole, linear = linearní pole, 0 = RAID 0, 1 = RAID 1, atd.

persistent-superblock: řeší problém když je nutno bootovat z RAID pole, bude vysvětleno dále, kap. 1.3.1.4.

chunk-size: definuje nejmenší „atomický“ kus dat [kb] který lze zapsat na disk, nutno volit dle uvážení podle stejných pravidel, jako se volí velikost clusteru. Pro malé soubory je lepší menší velikost (šetří se místo na disku) pro velké soubory je možno volit větší chunk size

nr-raid-disks: uvádí kolik disků máme v poli, resp. kolik diskových oddílů

nr-spare-disks: uvádí počet spare disků v poli, může být 0 až N

device: jméno zařízení vždy následované **raid-disk** (aktivní oddíl), **spare-disk** (záložní oddíl), **parity-disk** (paritní oddíl např. u RAID 3 a 4) a **fail-disk** označující selhaný disk, při inicializaci se přeskočí

parity-algorithm: uvádí použitý paritní algoritmus s RAID 3, 4, 5, k dispozici jsou: left-symmetric (obecně nejrychlejší), right-symmetric, right-asymmetric

- Vysvětlivky k mdadm

--raid-devices= specifikuje počet aktivních oddílů v poli

--spare-devices= specifikuje počet záložních oddílů v poli

--chunk= udává velikost v kilobytech po kolika se budou data rozdělovat mezi disky

--level= udává RAID level, možnosti jsou: linear, raid0, 0, stripe, raid1, 1, mirror, raid4, 4, raid5, 5, raid6, 6, raid10, 10, multipath, mp, faulty, některé položky jsou synonyma.

--verbose „ukecanější“ výstup programu

2.3.4 Mody mdadm

--create vytvoří pole

--build vytvoří pole bez superbloků

--assemble sestaví dříve existující pole

--follow --monitor vybere monitorovací mód

--grow mod pro manipulaci s pole jako je přidávání/odebírání disků a změny parametrů pole

další použití mdadm viz manuálová stránka [25].

2.3.5 RAID SuperBlock

každý aktivní oddíl pole obsahuje 128 kB vyhrazenou část pro informace o příslušnosti daného svazku ke konkrétnímu poli o stavu pole, atd. Pro RAID 0 a linear lze vytvořit pole bez superbloků (mdadm --build nebo persistent-superblock 0 u /etc/raidtab) v takovém případě je ale RAID svazek nebootovatený a při každém startu systému se musí pole znovu obnovit

2.3.6 Spuštění RAIDu při startu systému

Jednou možností je použití utilit raidstart a raidstop. Pomocí těchto utilit můžeme pole aktivovat či zastavit kdykoliv, stačí tedy upravit přísluné startovací skripty. (Pokud distribuce toto neobsahuje; např. distribuce Red Hat není potřeba upravovat, ze skriptu /etc/rc.d/rc.sysinit je raidstart volán automaticky, existuje-li soubor /etc/raidtab a raidstop je volán ze skriptu /etc/rc.d/init.d/halt). Druhou, robustnější metodou je využití možnosti automatické aktivace polí jádrem při bootu. Aby mohla fungovat, je potřeba používat perzistentní RAID superbloky a všechny diskové oddíly, které jsou součástí polí, musejí být v tabulce oddílů označeny jako typ Linux raid autodetect (hodnota 0xfd). Výhodou tohoto řešení navíc je, že jakmile je diskové pole inicializováno, nepoužívá se již pro opětovný start/zastavení pole konfigurační soubor /etc/raidtab. O sestavení a sputění pole se postará ovladač RAIDu, který na všech diskových oddílech typu Linux raid autodetect vyhledá RAID superbloky a na základě informací v RAID superblocích pole spustí. Stejně tak ovladač RAIDu

všechny pole korektně vypne v závěrečné fázi ukončení běhu systému poté, co jsou odpojeny souborové systémy. I v případě změny jmen disků nebo po přenesení disků na úplně jiný systém tedy bude pole korektně sestaveno a nastartováno.

2.3.7 Microsoft Windows Software RAID

Následující text je převzatý z literatury [5] a [8].

2.3.8 Microsoft Windows 2003 Server

V prostředí MS Windows 2003 server je postup velmi jednoduchý, systém primárně RAID podporuje, Windows XP bohužel ne. Vytvoření pole lze provést grafickou cestou nebo pomocí utility příkazového řádku diskpart.

2.3.9 RAID 1 a Utilita diskpart

Předpokládáme, že máme dva disky, disk 0 a disk 1 ze kterých chceme vytvořit RAID pole typu 1, disk 0 je systémový, disk 1 je stejně velký, nenaformátovaný, bez oddílů.

1) změníme disk 0 (systémový) na dynamický, v příkazové řádce:

```
diskpart
list disk
select disk 0
convert dinamic
```

2) To samé provedeme s druhým diskem:

```
diskpart
list disk
select disk 1
convert dinamic
```

3) ve správci disků zvolíme u disku 0 „Přidat zrcadlo“ a pokračujeme podle pokynů průvodce (bude podrobněji popsáno u Windows XP).

4) Při tomto postupu se na druhý disk nepřenese zaváděcí oblast disku, MBR (Master Boot Record), z druhého disku by tudíž nešlo bootovat i když data na něm obsažená jsou plnou kopií dat na disku prvním, aby byly disky na 100 % stejné MBR

překopírujeme i na druhý disk. Nejjednodušší způsob jak toto provést je pomocí LiveCD některé Linuxové distribuce a provedení příkazu:

```
#dd if=/dev/sda of=/dev/sdb bs=512 count=1
```

Což zkopíruje prvních fyzických 512 B prvního disku na disk druhý, tudíž i MBR, kde se tato zaváděcí oblast vždy nalézá, respektive její první stádium.

2.3.10 Microsoft Windows XP SP2

Jak již bylo naznačeno Windows XP software RAID primárně nepodporuje, nástroje na jeho vytvoření jsou však v systému obsažené. Musíme tedy provést mírnou úpravu tří systémových souborů (dmadmin.exe, dmconfig.dll a dmboot.sys) aby nám systém RAID dovolil vytvořit, jinak jsou v grafickém prostředí položky důležité pro vytvoření pole nepřístupné (vyšeděné).

1) Upravíme potřebné soubory v hexadecimálním editoru (např. PSPad)

dmadmin.exe (typicky v C:\Windows\Sysytem32\dmadmin.exe)

originál

```
00001c30  7365  7276  6572  6E74  0000  0000  6C61  6E6D
znaková reprezentace hexa hodnot: servernt….lanm
00001c40  616E  6E74  0000  0000  5072  6F64  7563  7454
znaková reprezentace hexa hodnot: annt….ProductT
```

upravený

```
00001c30  7769  6E6E  7400  0000  0000  0000  6C61  6E6D
znaková reprezentace hexa hodnot: winnt…….lanm
00001c40  616E  6E74  0000  0000  5072  6F64  7563  7454
znaková reprezentace hexa hodnot: annt….ProductT
```

dmconfig.dll (typicky v C:\Windows\Sysytem32\dmconfig.dll)

originál

```
00005140  4C41  4E4d  414E  4E54  0000  0000  5345  5256
znaková reprezentace hexa hodnot: LANMANNT….SERV
00005150  4552  4E54  0000  0000  5749  4E4E  5400  0000
znaková reprezentace hexa hodnot: ERNT….WINNT…
```

upravený

```
00005140  4c41  4E4D  414E  4E54  0000  0000  5749  4E4E
znaková reprezentace hexa hodnot: LANMANNT….WINN
00005150  5400  0000  0000  0000  5345  5256  4552  4E54
znaková reprezentace hexa hodnot: T…….SERVERNT
```

dmboot.sys (typicky v C:\Windows\Sysytem32\drivers\dmboot.sys)

originál

```
00011070  5400  7900  7000  6500  0000  5749  4E4E  5400
znaková reprezentace hexa hodnot: T.y.p.e…WINNT.
00011080  0000  5345  5256  4552  4E54  0000  0000  4C41
znaková reprezentace hexa hodnot: ..SERVERNT….LA
```

upravený

```
00011070  5400  7900  7000  6500  0000  5345  5256  4552
znaková reprezentace hexa hodnot: T.y.p.e…SERVER
00011080  4E54  5749  4E4E  5400  0000  0000  0000  4C41
znaková reprezentace hexa hodnot: NTWINNT…….LA
```

Jak je z překladu hexadecimálních čísel na ASCII znaky vidět, jedná se pouze o změnu řetězce „WINNT“ na „SERVERNT“ a obráceně.

2) Převedeme disky které chceme mít v RAID poli na dynamické, postup stejný jako u Win 2003 nebo pomocí grafického správce disků (diskmgmt.msc)

3) Upravené soubory je třeba zkopírovat na jejich správné umístění v systému když je systém vypnut. To lze provést buď:

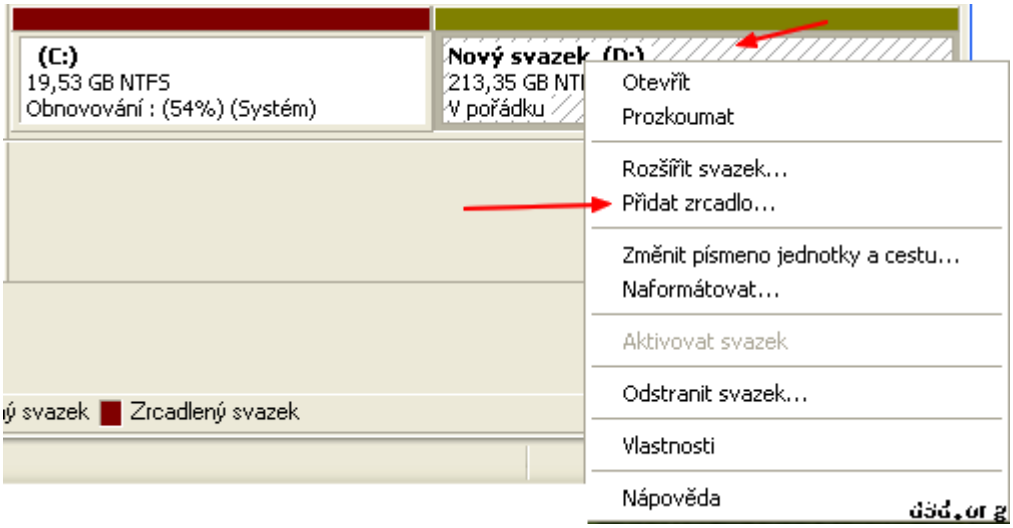
3.1) Zkopírováním souborů pomocí konzole pro zotavení, která se nachází na instalační CD systému: vložíme CD s instal. Windows XP do CD-Romu. Nabootujeme z CD-Rom a spustíme instalaci Windows XP po zavedení použijeme „konzole pro zotavení (R)“ připojíme se k systému jako administrátor a nakopírujeme změněné soubory do adresáře C:\Windows\system32\.

```
copy c:\Windows\_raid\dmboot.sys system32\drivers
copy c:\Windows\_raid\dmboot.sys system32\dllcache
copy c:\Windows\_raid\dmconfig.dll system32
```

```
copy c:\Windows\_raid\dmconfig.dll system32\dllcache
copy c:\Windows\_raid\dmadmin.exe system32
copy c:\Windows\_raid\dmadmin.exe system32\dllcache
```

3.2) Použijeme LiveCD[2] Linuxové distribuce které podporuje souborový systém od Microsoftu a zkopírujeme upravené soubory do stejného umístění jako v případě 3.1.

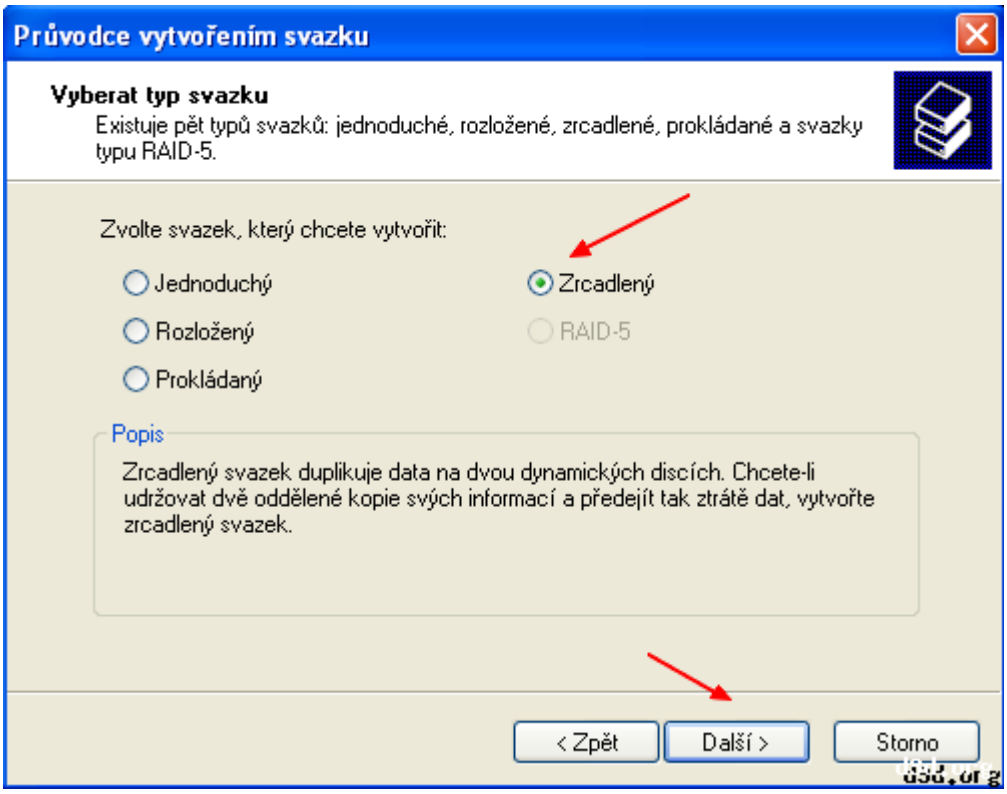
4) Restartujeme, necháme naběhnout systém klasickým způsobem a zapneme zrcadlení disků ve správci disků, na disku 0 vybereme přidat zrcadlo (ilustrace 7).



Ilustrace 8: Windows SW RAID přidání zrcadla

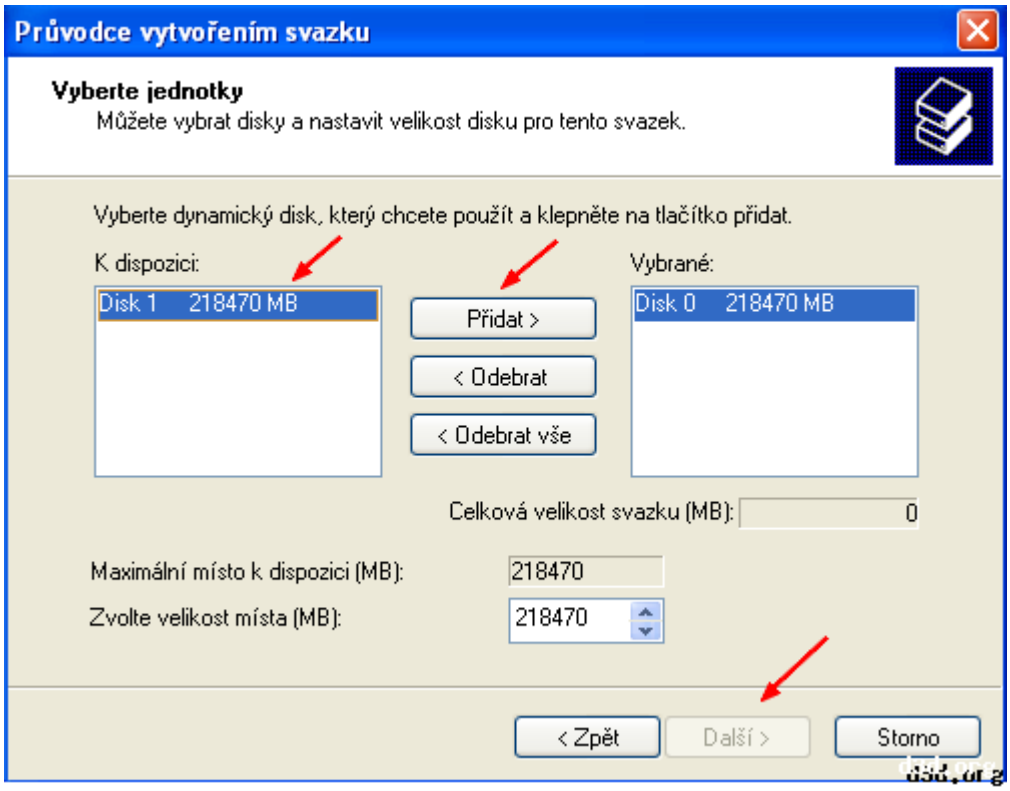
5) Vybereme, že chceme zrcadlený, tedy RAID 1 (ilustrace 8).

2 Distribuce operačního systému sestavená tak, že systém zle zavést z CD-ROM aniž by byl nainstalován na pevném disku



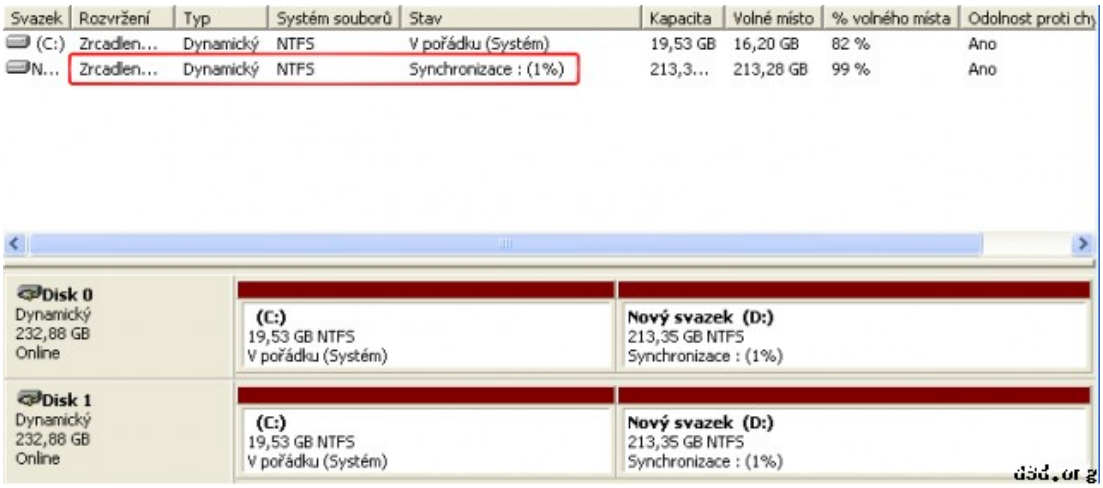
Ilustrace 9: Windows SW RAID výběr typu RAID levelu

6) přidáme k disku 0 disk 1 (zde můžeme přidat i více než jeden) ilustrace 10.



Ilustrace 10: Windows SW RAID průvodce vytvoření svazku

7) počkáme než systém provede synchronizaci (ilustrace 11).



Ilustrace 11: Windows SW RAID synchronizace

2.4 Hardware RAID

Informace o tomto tématu jsem čerpal z zdroje [9], [10], uživatelské příručky k základní desce Asus A8V a elektronické příručky k RAID řadičům VIA8238 a FastTrak378.

2.4.1 RAID pomocí řadiče disku

RAID vytvořený pomocí řadiče disku je považován za ten jediný a správný hardware raid, myslí se tím provádění RAID operací přímo elektronickými obvody řadiče, nikoliv jakékoli předávání výpočtů hlavnímu procesoru nebo jiným výpočetním jednotkách počítače.

2.4.2 Historie

Hardware RAID řadiče disků již existují velmi dlouho, ovšem vždy vyžadovaly drahé SCSI (Small Computer System Interface) pevné disky a byly zaměřené na serverové použití nebo do špičkových konfigurací pro domácí počítače. Výhody SCSI technologie bylo až 15 zařízení na jedné sběrnici, nezávislý přenos dat, připojování a odpojování zařízení za běhu a vysoké hodnoty času mezi selháními (MTBF[3]). Okolo roku 1997 bylo představeno diskové rozhraní ATAPI-4, více známé

3 Mean Time Between Failures – střední doba mezi selháními, více viz [11]

jako Ultra ATA/33 které přineslo nový typ přenosu dat nazvaný Ultra DMA[4] Mode 0, který umožňoval rychlejší přenos dat s využitím menšího množství procesorového času. Společně s tímto byl představen i první ATA[5] RAID řadič v provedení jako PCI[6] rozšiřující karta. Protože RAID systémy mají tendenci k tomu obsahovat hodně disků cenová výhoda ATA disků umožňovala stavět RAID systémy za mnohem menší náklady než s použitím SCSI. Díky tomu se RAID technologie dostala i mezi běžné uživatele žádající ochranu proti selhání disků bez investování do drahých SCSI disků. Od počátku 21. století každá vybavenější základní deska obsahuje alespoň jednoduchý ATA nebo SATA[7] RAID řadič.

2.4.3 Teorie

Hardware řadiče disků používají většinou, výrobce od výrobce, jiné rozvržení dat na disku, tudíž je často nemožné použít na RAID-ovém diskovém poli řadič od jiného výrobce než na řadiči na kterém bylo pole vytvořeno. Hardware řadiče nespotřebovávají čas procesoru ani jiné systémové zdroje, BIOS[8] z něj může zavést systém a větší integrace s ovladačem řadiče může poskytnout lepší ošetření případných chyb.

Hardware implementace RAIDu vyžaduje minimálně speciální řadič disků. Na stolním počítači to může být PCI rozšiřující karta, PCI-E rozšiřující karta nebo může být řadič vestavěný na základní desce. Mohou být použity řadiče podporující většinu typů diskových rozhraní, jako IDE/ATA, SATA, SCSI, SSA[9], FC[10], někdy i kombinace několika. Řadiče mohou být v nezávislých zařízeních, např. externí rámeček na disk, který může být připojen k počítači přímo nebo přes SAN[11] nebo mnohem častěji

4 Direct Memory Access – technika přístupu k paměti bez použití hlavního procesoru, více [17]
5 Advanced Technology Attachment – postarší rozhraní řadičů disků, více viz [18]
6 Peripheral Component Interconnect - rozhraní v současné době používané na základních deskách pro připojování rozšiřujících karet, více viz [19]
7 Serial Advanced Technology Attachment – hojně používané rozhraní řadičů disků, více viz [20]
8 Basic Input Output System – základní vstupně výstupní systém, obsahuje ho každá základní deska, více viz [21]
9 Serial Storage Architecture – sériový transportní protokol používaný na serverech, vyvinutý v r. 1990 firmou IBM, více viz [13]
10 Fibre Channel – sítová technologie o rychlosti 1-12 gigabitů primárně používaná pro síťová úložiště, více viz [14]
11 Storage Area Network – síťová architektura k připojování síťových diskových polí k počítačům takovým způsobem, že se diskové pole operačnímu systému jeví jako přímo připojené, více viz [15]

součástí počítače. Samotný hardware řadič se stará o správu jednotek a provádí všechny kalkulace parit apod. vyžadovaných konkrétním RAID levelem.

Většina hardware implementací poskytuje také čtecí a zapisovací vyrovnávací paměť, která v závislosti na množství vstupně výstupních operací může zvýšit výkon celého pole. V mnoha systémech je zapisovací vyrovnávací paměť podpořená bateriovým napájením, aby v případě výpadku napájení nebyla ztracena žádná data obsažená ve vyrovnávací paměti, která ještě nebyla zapsána na disky.

Hardware implementace také často podporuje přidávání nebo odebírání za chodu, to je ovšem hlavně závislé na použitém rozhraní než na schopnostech řadiče.

2.4.4 Hardware RAID řadič VIA VT8237

Tato kapitola se zabývá popisem RAID řadiče, integrovaného v jižním můstku základní desky Asus A8V, VIA VT8237.

Tento řadič podporuje tři RAID levely, a to: RAID 0, 1 a JBOD což je velmi obvyklá konfigurace u těchto levných integrovaných řešení.

2.4.5 Klíčové vlastnosti řadiče VIA VT8237

- Podporuje dva SATA pevné disky.
- Podporuje pevné disky rozhraní SATA 1. generace.
- Podporuje disky větší než 137GB (48-bit LBA[12]).
- Duální nezávislé ATA kanály s maximem dvou připojených disků.
- Podporuje SATA, PIO[13] a DMA.
- Podporuje PCI Plug and Play[14], sdílení přerušení a koexistování s řadičem základní desky.
- Podporuje RAID 0, 1 a JBOD.

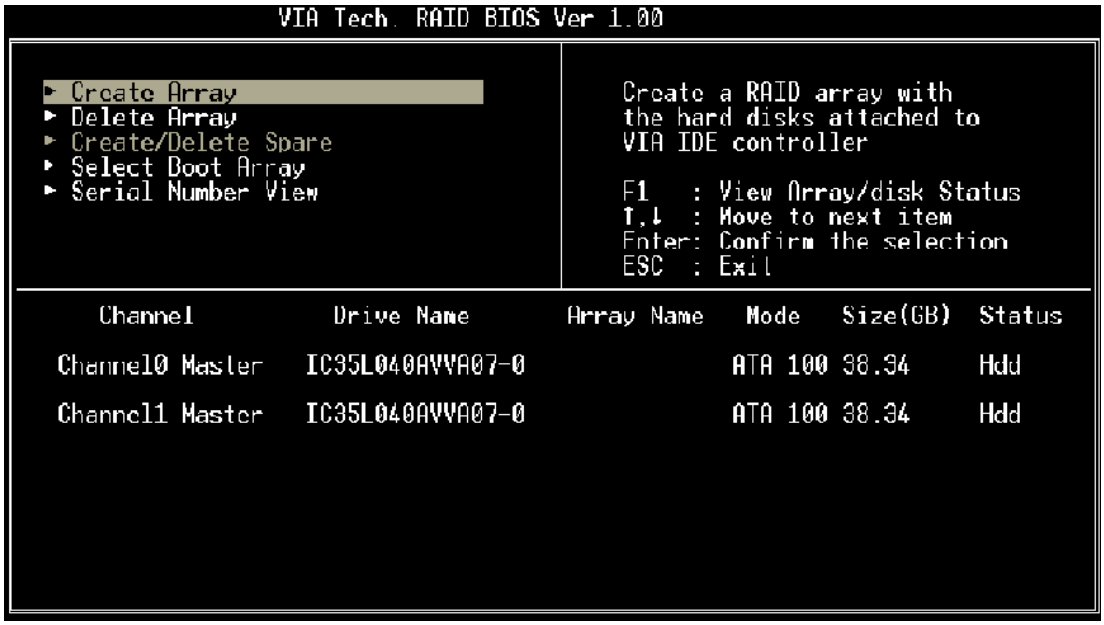
12 Logical Block Addressing – schéma jak specifikovat umístění bloku dat na počítačovém záznamovém mediu, typicky sekundární zařízení jako jsou pevné disky, více viz [16]

13 Programmed Input/Output – metoda pro přesun dat mezi hlavním procesorem a periferiemi, jako jsou síťové adaptéry nebo ATA disky

14 Vlastnost počítačových komponentů, kdy přidání nového zařízení typicky periferie nevyžaduje žádnou konfiguraci nebo manuální instalaci ovladačů, více viz [22]

- 4 kB až 64 kB blok při prokládaní.
- RAID konfigurační a řídící software nástroj kompatibilní s BIOSem pro platformu Windows.
- Monitorování stavu zařízení a ohlášení chyby v reálném čase.
- Podporuje odpojení za běhu poškozeného SATA disku v RAID 1 poli.
- Podpora automatického obnovení zrcadlených disků.
- Podporuje ATA SMART[15].
- Podpora MS 98, ME, NT4.0, 2000, XT operačních systémů.
- Žurnál událostí pro snadné odstraňování problémů.
- On-line pomoc pro snadné používání RAID software.

2.4.6 BIOS konfigurační utilita



Ilustrace 12: VIA Tech. RAID BIOS utilita

Konfigurační utilita poskytuje následující operace nad diskovým polem:

- vytvořit diskové pole

15 Self-Monitoring, Analysis, and Reporting Technology – monitorovací systém pro počítačové pevné disky, více viz [23]

Pro vytvoření diskového pole můžeme vybrat buď automatické nastavení které udělá vše za nás, do pole vloží jednoduše všechny disky připojené k řadiči, my vybereme jen typ RAIDu. Pokud chceme vše dělat ručně musíme nejdříve vybrat RAID level, který chceme použít, poté vybrat disky, které chceme v poli mít, poté volitelně pokud jsme vybrali RAID level 0 musíme ještě vybrat velikost bloku po kterých se budou data na discích dělit, volba je 4 kB, 8 kB, 16 kB, 32 kB a 64 kB. Poté stačí vybrat „Start create process“ který započne s vytvářením pole, po vytvoření budou všechna data, která na discích byla předtím ztracena.

- smazat diskové pole

Zde jednoduše vybereme existující diskové pole a necháme ho smazat, toto opět znehodnotí všechna data na discích kromě disků z RAIDu 1.

- vybrat pole ze kterého se bude zavádět systém

Zde můžeme vybrat který z RAID polí má mít nastavený příznak boot.

- výpis stavu pole

Zde při stisku klávesy F1 nám utilita vypíše informace o RAID poli, pokud je nějaké nakonfigurované.

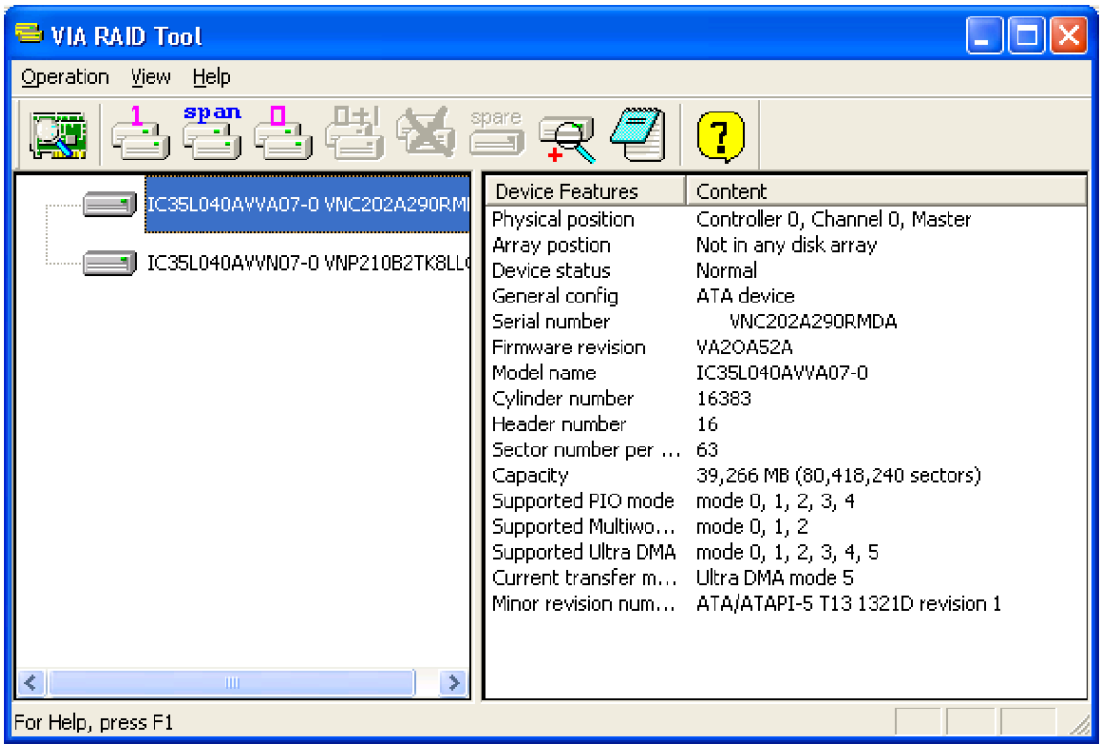
- duplikace kritického RAID 1 pole

Tato funkce se zaktivuje pouze v případě pokud při bootu BIOS zjistí nekonzistenci mezi uživatelským datovým diskem (source) a záložním (mirror). Nabízí buďto duplikovat disk nebo nedělat nic a pokračovat v bootu.

- oprava poškozeného RAID 1 pole

Tato funkce se aktivuje pouze při bootu pokud BIOS zjistí jeden nebo více disků z RAID pole levelu 1 je nefunkční nebo úplně chybí. Nabízí nám možnost vypnout pole a zkontrolovat chybný disk nebo rozbít zrcadlený vztah nebo vybrat náhradní (spare) disk a provést obnovení nebo nedělat nic a pokračovat v bootu, tato volba také umožňuje provést duplikaci na záložní disk až z operačního systému.

2.4.7 Windows konfigurační utilita



Ilustrace 13: VIA RAID Tool pro Windows

Konfigurační utilita pro prostředí Windows poskytuje nejen líbivější konfigurátor než je jeho BIOS verze protějšek, ale hlavně možnost monitorovat RAID pole v reálném čase a přístup k žurnálu událostí. To co dělá BOIS utilita při bootu systému, tedy kontrolu konzistence a kontrolu stavu disků z pohledu chyb a jejich přítomnosti v systému, tento program dělá průběžně.

Ovšem dříve než můžeme toto použít musíme nainstalovat VIA RAID ovladače tohoto řadiče. Postup je ovšem v tradici Windows velmi jednoduchý, sestávající ze spuštění setup.exe na instalačním CD, odklikání instalačního pomocníka a konečném restartu systému.

2.5 Hardware RAID pomocí ovladačů nebo firmware řadiče

RAID založený na operačním (software RAID) systému nemůže bezpečně ochránit zaváděcí proces a je obecně nepraktický na desktopových verzích operačního systému Windows, jak bylo popsáno výše. Hardware RAID řadiče jsou drahé. K vyplnění této mezery byli využity levné „RAID řadiče“ které sami neobsahují čip

RAID řadiče, ale pouze standardní čip řadiče se speciálními ovladači a programovým vybavením (firmware). Během ranných stádií startu počítače RAID funkci řídí programové vybavení čipu řadiče, jakmile je zaveden operační systém ve chráněném módu, jako například moderní verze GNU/Linuxu nebo Microsoft Windows, přebere řízení ovladač (driver).

Tyto řadiče jsou jejich výrobci popisovány jako RAID řadiče a pro spotřebitele je velmi málo kdy jasné, že tíhu RAID operací má na starosti hlavní procesorová jednotka, ne RAID řadič sám. Ještě před uvedením těchto „RAID řadičů“ bylo naznačeno, že řadič sám bude provádět potřebné výpočty. Díky tomu vešli ve známost jako „falešný RAID“ (fake RAID) i přes to, že samotné RAID pole řadiče implementují korektně.

3 Reed-Solomonovy kódy

Tato kapitola popisuje teorii Reed-Solomonových kódů jejich hlavní vlastnosti a základy toho jak fungují, hlavní zdroje ze kterých jsem čerpal jsou [27] a [28].

3.1 Úvod

V roce 1960 Irving S. Reed a Gustave Solomon vydali článek v „Journal of the Society for Industrial and Applied Mathematics“ [29]. Tento článek popisoval novou třídu opravných kódů, která se dnes nazývá Reed Solomonovy (R-S) kódy. Tyto kódy jsou velmi silné a mnohostranné, v dnešní době se používají v mnoha odvětvích od kompaktních disků, po aplikace v hlubokém kosmu, jako součást signálů při komunikaci s prvními sondami Voyager.

Dnes nejrozšířenější použití Reed-Solomonových kódů zahrnuje opravné kódy na CD/DVD/Blue Ray diskových nosičích, opravné kódy v přenosu pozemního a satelitního digitálního vysílání a spousta dalších aplikací přenosu dat při kterých dochází k chybám během transmise.

Reed-Solomonovy kódy jsou nebinární cyklické kódy se symbolovými sekvencemi o m bitech, kde m je jakékoli celé kladné číslo mající hodnotu větší než R-S (n, k) kód kde na m-tém bitu platí pro všechna n a k následující:

$$0 < k < n < 2^m + 2$$
rovnice 3.1

Kde k je počet datových symbolů, které se zakódovávají a n je celkový počet symbolů v celém zakódovaném bloku. Pro konvenční R-S (n, k) platí, že:

$$(n, k) = (2^m - 1, 2^m - 1 - 2t)$$
rovnice 3.2

Kde t je symbolová schopnost opravy chyb a $n - k = 2t$ je počet paritních symbolů. Rozšířený R-S kód může být použit s $n = 2^m$ nebo $n = 2^m + 1$, ale už ne více.

Reed-Solomonovy kódy dosahují největší možná hodnoty takzvané minimální vzdálenosti ze všech lineárních kódů se stejným blokem dat na vstupu enkodéru a výstupu z něj. Pro nebinární kódy je dána vzdálenost mezi dvěma kódovými slovy (podobně jako u Hammingovy vzdálenosti), jako počet symbolů ve kterých se sekvence liší. Pro Reed-Solomonovy kódy je minimální vzdálenost kódu dána následující poučkou.

$$d_{min} = n - k - 1$$
rovnice 3.3

Kód je schopný opravit jakoukoli kombinaci chyb o počtu t nebo menším, kde se t dá vyjádřit jako:

$$t = \left[\frac{d_{min} - 1}{2}\right] = \left[\frac{n - k}{2}\right]$$
rovnice 3.4

Rovnice (4) ilustruje, že v případě R-S kódů, oprava t chybných symbolů nevyžaduje více než $2t$ paritních symbolů. Rovnice (4) sama vede k následující myšlence. Dalo bys se říci, že dekodér má $n - k$ redundantních symbolů, které může „spotřebovat“, což je dvojnásobek počtu opravitelných chyb. Pro každou chybu je použit jeden redundantní symbol k nalezení chyby a jiný redundantní symbol je použit k nalezení správné hodnoty.

Opravná schopnost kódu, ρ, se dá vyjádřit jako:

$$\rho = d_{min} - 1 = n - k$$
rovnice 3.5

Simultánní oprava chyb a schopnost opravy chyb se dá vyjádřit jako:

$$2\alpha + \gamma < d_{min} < n - k$$
rovnice 3.6

Kde α je je počet chybných symbolů, které se dají opravit a γ je počet vymazaných symbolů, které se dají opravit. Výhoda nebinárních kódů, jako jsou Reed-Solomonovy, lze vidět v následujícím porovnání. Předpokládejme binární (*n, k*) = (7, 3) kód. Celý *n* tý prostor obsahuje $2^n = 2^7 = 128$ *n* tin, z kterých $2^k = 2^3 = 8$ (nebo 1/16 n tin) jsou kódová slova. Dále, předpokládejme nebinární (*n, k*) = (7, 3) kód, kde každý symbol je sestaven z *m* = 3 bitů, *n* tý prostor obsahuje $2^{nm} = 2^{21} = 2{,}097{,}152$ *n* tin, ze kterých jsou $2^{km} = 2^9 = 512$ (nebo 1/4096 *n*-tin) kódová slova. Tento poměr se snižuje se zvyšující se hodnotou *m*. Důležitá věc zde je, že když malá část *n* tého prostoru je použita pro kódové slovo, může být vytvořeno velké d_{min}.

Jakýkoli lineární kód je schopen opravit *n* - *k* smazaných symbolů pokud všech *n* - *k* smazaných symbolů leží v oblasti paritních symbolů. Nicméně, R-S kódy mají tu úžasnou vlastnost, že jsou schopny opravit jakýkoli set *n* - *k* smazaných symbolů kdekoli v bloku dat. R-S kódy mohu být navrženy aby měli jakoukoli redundanci. Nicméně složitost vysoko rychlostních implementací roste s redundancí, proto jsou nejpoužívanější R-S kódy s vysokou rychlostí kódování a tedy nízkou redundancí.

3.2 Konečná pole

Abychom pochopili kódovací a dekódovací principy nebinárních kódů, jako jsou Reed-Solomonovy kódy, je třeba nahlédnout do oblasti konečných polí známých jako Galoisovo pole (GF^{16}). Pro jakékoli prvočíslo, *p*, existuje konečné pole označené GF(*p*), které obsahuje *p* elementů. Je možné rozšířit GF(*p*) na pole p^m elementů zvané rozšiřující pole GF(*p*) a psané jako GF(p^m), kde *m* je nezáporné celé číslo.

16 Galois Field – Galoisovo pole pojem z abstraktní algebry, pojmenováno podle Évariste Galoise [30]

Všimněme si, že GF(p^m) obsahuje podmnožinu elementů GF(p). Symboly z rozšířeného pole GF(2^m) se používají pro konstrukci Reed-Solomonových kódů.

Binární pole GF(2) je podmnožinou rozšířeného pole GF(2^m) v úplně stejném smyslu jako jsou reálná čísla podmnožinou čísel komplexních.

Mimo jiné čísla 0 a 1, která jsou unikátními elementy v rozšířeném poli budou reprezentovány jako nový symbol α. Každý nenulový element pole GF(2^m) může být reprezentován jako mocnina α. Nekonečná množina elementů, F, je na počátku formována elementy {0, 1, α} a generuje další elementy progresivním násobením poslední hodnoty α, což popisuje následující:

$$F=\{0,1,\alpha,\alpha^2,\ldots,\alpha^j,\ldots\}=\{0,\alpha^0,\alpha^1,\alpha^2,\ldots,\alpha^j,\ldots\}$$

rovnice 3.7

K získání konečné množiny elementů GF(2^m) z *F,* musí být na *F* zavedena taková podmínka, aby obsahovalo pouze 2^m bylo uzavřené pod násobením. Podmínka, která uzavírá množinu elementů pod násobením je charakterizována nerozložitelným polynomem jako je naznačen níže:

$$\alpha^{2^m-1}+1=0$$

rovnice 3.8

Nebo ekvivalent:

$$\alpha^{2^m-1}+1=\alpha^0$$

rovnice 3.9

Použitím tohoto omezení polynomů jakýkoli element pole, který má mocninu stejnou nebo větší než 2^m -1 může být rozdělen na elementy s mocninou menší než 2^m -1, tak jako v následující ukázce:

$$\alpha^{2^m+n}=\alpha^{2^m+1}\alpha^{n+1}=\alpha^{n+1}$$

rovnice 3.10

Poté rovnice (10) může být použita na zformování konečné sekvence *F** z nekonečné sekvence *F* jako v následujícím příkladu:

$$F = \left\{0, 1, \alpha, \alpha^2, \ldots, \alpha^{2^m-2}, \alpha^{2^m-1}, \alpha^{2^m}, \ldots\right\}$$
$$F = \left\{0, \alpha^0, \alpha^1, \alpha^2, \ldots, \alpha^{2^m-2}, \alpha^0, \alpha^1, \alpha^2, \ldots\right\}$$
rovnice 3.11

Tedy z rovnice (12) můžeme vidět, že elementy konečného pole GF(2^m) jsou následující:

$$GF(2^m) = \left\{0, \alpha^0, \alpha^1, \alpha^2, \ldots, \alpha^{2^m-2}, \alpha^0, \alpha^1, \alpha^2\right\}$$
rovnice 3.12

3.3 Sčítání a násobení v rozšířeném poli GF(2^m)

Každý z 2^m elementů konečného pole GF(2^m) může být reprezentován jako jedinečný polynom stupně m - 1 nebo méně. Stupeň polynomu závisí na hodnotě jeho nejvyššího exponentu. Každý nenulový element GF(2^m) vyjádříme jako polynom $a_i(X)$, kde alespoň jeden z m elementů je nenulový. Pro i = 0,1,2,...,2m - 2 je polynom tvořen takto:

$$\alpha^i = a_i(X) = a_{i,0} + a_{i,1}X + a_{i,2}X^2 + \ldots + a_{i,m-1}X^{m-1}$$
rovnice 3.13

Berme v úvahu případ, kdy m = 3, konečné pole je tedy pole GF(2^3). Tabulka 2 popisuje mapování sedmi elementů $\{\alpha^i\}$ a nulového elementu, podle pravidel základních elementů popsaných v rovnici 3.12. Díky rovnici 3.9 víme, že $\alpha^0 = \alpha^7$, proto je zde sedm nenulových elementů a osm elementů v poli celkem. Každý řádek v tabulce zahrnuje sekvenci binárních hodnot reprezentujících koecifienty $a_{i,0}$,$a_{i,1}$ a $a_{i,2}$ v rovnici . Jedna z výhod použití elementů $\{\alpha^i\}$ rozšířeného pole je, že na místo binárních elementů je zde kompaktní notace odrážející matematickou reprezentaci nebinárního kódovacího a dekódovacího procesu. Součet dvou elementů v konečném poli je reprezentován jako modulo 2 součtu elementů se stejnou mocninou to znamená že možný výsledek je buď 0 nebo původní sčítanec, např.: $\alpha^3 + \alpha^3 = 2$ modulo 2 = 0 nebo $\alpha^3 + \alpha^3 + \alpha^3 = 3$ modulo 2 = $\alpha^3 = \alpha + 1$, v binární reprezentaci je to operace XOR mezi konkrétními elementy, tedy podle předchozího příkladu:

110 XOR 110 = 0

nebo

110 XOR 110 XOR 110 =

110 XOR 000 = 110

Kdybychom α^3 ještě rozdělili podle $\alpha^3 = \alpha + 1$ postup by byl následující:

$\alpha + 1 + \alpha + 1 + \alpha + 1 = \alpha + 1 = \alpha^3$

Výsledek je stejný, binární reprezentace a exkluzivní součet:

010 XOR 100 XOR 010 XOR 100 XOR 010 XOR 100 =

110 XOR010 XOR 100 XOR 010 XOR 100 =

100 XOR 100 XOR 010 XOR 100 =

000 XOR 010 XOR 100. =

010 XOR 100 = 110 = α^3

Výsledek je opět stejný. Sčítání elementů s nestejnými mocninami je totožný, mocniny se rozloží na tři základní α^0, α^1, a α^2 elementy, spočítá se jejich součet ve výrazu a podle sudosti nebo lichosti součtu se určí které sčítance ve výsledku mají zůstat a které ne.

Tabulka 3.1: GF(2^3)

Elementy pole	Základní elementy		
	X^0	X^1	X^2
0	0	0	0
α^0	1	0	0
α^1	0	1	0
α^2	0	0	1
α^3	1	1	0
α^4	0	1	1
α^5	1	1	1
α^6	1	0	1
α^7	1	0	0

3.4 Rozšířené pole GF(2^3)

Vezměme v úvahu příklad obsahující primitivní polynom a konečné pole, které je jím popsáno. V tabulce 3.3 je seznam některých nedělitelných primitivních polynomů. Vybereme si ten z prvního řádku, $f(X) = 1 + X + X^3$, který popisuje konečné pole GF(2^m) kde stupeň polynomu je $m = 3$. Je v něm tedy $2^m = 2^3 = 8$ elementů v poli které je popsané pomocí $f(X)$. Řešení jak nalézt kořeny funkce $f(X)$ je najít hodnoty X, kde se funkce rovná nule, $f(X) = 0$. Již známé binární elementy 1 a 0 nám nestačí (nejsou kořeny) polynomu $f(X) = 1 + X + X^3$, protože $f(1) = 1$ a $f(0) = 1$. Nicméně základní teorém algebry velí, že polynom stupně m musí mít *přesně m* kořenů. Proto v tomto případě $f(X) = 0$ musí poskytnout tři kořeny. Nastává zde dilema když všechny tři kořeny neleží ve stejném konečném poli jako koeficienty $f(X)$. Poté musí ležet někde jinde a to v rozšířeném poli GF(2^3). Nechť je α element rozšířeného pole a je definován jako kořen polynomu $f(X)$. Poté je možno psát následující:

$$f(\alpha)=0$$
$$1+\alpha+\alpha^3=0$$
$$\alpha^3=-1-\alpha$$

rovnice 3.14

Protože v binárním poli je -1 totožné s +1, -1 = +1, α^3 může být psán jako:

$$\alpha^3=1+\alpha$$

rovnice 3.15

Zde α^3 je vyjádřen jako vážený součet α elementů nižších řádů. V skutečnosti všechny mocniny α mohou být takto vyjádřeny. Například, uvažujme α^4 když použijeme vyjádření z předchozí rovnice a přidáme další element α dostaneme:

$$\alpha^4=\alpha\cdot\alpha^3=\alpha\cdot(1+\alpha)=\alpha+\alpha^2$$

rovnice 3.16

Nyní stejným způsobem vytvořme α^5:

$$\alpha^5=\alpha\cdot\alpha^4=\alpha\cdot(\alpha+\alpha^2)=\alpha^2+\alpha^3$$

rovnice 3.17

Když dosadíme z rovnice (3.15) pak:

$$\alpha^5 = 1 + \alpha + \alpha^2$$
rovnice 3.18

Nyní pro α^6 použitím předešlé rovnice dostaneme:

$$\alpha^6 = \alpha \cdot \alpha^5 = \alpha \cdot (1 + \alpha + \alpha^2) = \alpha + \alpha^2 + \alpha^3 = 1 + \alpha$$
rovnice 3.19

A nakonec α^7 použitím předešlé rovnice:

$$\alpha^7 = \alpha \cdot \alpha^6 = \alpha \cdot (1 + \alpha^2) = \alpha + \alpha^3 = \alpha^0$$
rovnice 3.20

Všimněme si, že $\alpha^7 = \alpha^0$ a proto osm elementů konečného pole GF(2^3) je:

$$\{0, \alpha^0, \alpha^1, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6\}$$
rovnice 3.21

Tabulka 3.2: Nedělitelné primitivní polynomy o řádu 3 – 24

m	Nedělitelný primitivní polynom	*m*	Nedělitelný primitivní polynom
3	$1 + X + X^3$	14	$1 + X + X^6 + X^{10} + X^{14}$
4	$1 + X + X^4$	15	$1 + X + X^{15}$
5	$1 + X^2 + X^5$	16	$1 + X + X^3 + X^{12} + X^{16}$
6	$1 + X + X^6$	17	$1 + X^3 + X^{17}$
7	$1 + X^3 + X^7$	18	$1 + X^7 + X^{18}$
8	$1 + X^2 + X^3 + X^4 + X^8$	19	$1 + X + X^2 + X^5 + X^{19}$
9	$1 + X^4 + X^9$	20	$1 + X^3 + X^{20}$
10	$1 + X^3 + X^{10}$	21	$1 + X^2 + X^{21}$
11	$1 + X^2 + X^{11}$	22	$1 + X + X^{22}$
12	$1 + X^1 + X^4 + X^6 + X^{12}$	23	$1 + X^5 + X^{23}$
13	$1 + X + X^3 + X^4 + X^{13}$	24	$1 + X + X^2 + X^7 + X^{24}$

Násobení je provádí stejnou procedurou jako sčítání, výsledek je součet exponentů modulo (2^m - 1) pro případ výše uvedeného příkladu to je 2^3 -1 tedy modulo 7

Následující tabulky 3.4 a 3.1 ukazuje všechny kombinace sčítanců a násobitelů z GF(2^3).

Tabulka 3.3: sčítání nad GF(23)

	α^0	α^1	α^2	α^3	α^4	α^5	α^6
α^0	0	α^3	α^6	α^1	α^5	α^4	α^2
α^1	α^3	0	α^4	α^0	α^2	α^6	α^5
α^2	α^6	α^4	0	α^5	α^1	α^3	α^0
α^3	α^1	α^0	α^5	0	α^6	α^2	α^4
α^4	α^5	α^2	α^1	α^6	0	α^0	α^3
α^5	α^4	α^6	α^3	α^2	α^0	0	α^1
α^6	α^2	α^5	α^0	α^4	α^3	α^1	0

Tabulka 3.4: násobení nad GF(23)

	α^0	α^1	α^2	α^3	α^4	α^5	α^6
α^0	α^0	α^1	α^2	α^3	α^4	α^5	α^6
α^1	α^1	α^2	α^3	α^4	α^5	α^6	α^0
α^2	α^2	α^3	α^4	α^5	α^6	α^0	α^1
α^3	α^3	α^4	α^5	α^6	α^0	α^1	α^2
α^4	α^4	α^5	α^6	α^0	α^1	α^2	α^3
α^5	α^5	α^6	α^0	α^1	α^2	α^3	α^4
α^6	α^6	α^0	α^1	α^2	α^3	α^4	α^5

3.5 Test primitivnosti polynomu

Jedna z cest jak poznat, že konkrétní nedělitelný polynom je také primitivním polynomem je díky předešlým znalostem celkem snadná. Aby mohl být nedělitelný polynom zároveň primitivním polynomem musí alespoň jeden z jeho kořenů být primitivní element. Pro připomenutí primitivní element je takový element, jehož mocnina se nachází v základním nebo rozšířené poli. Protože je pole konečné takovýchto elementů je omezený počet.

Máme příklad, kde chceme nalézt $m = 3$ kořenů polynomu $f(X) = 1 + X + X^3$ a ověřit zda takový polynom je primitivní kontrolou jestli alespoň jeden z kořenů je primitivní element.

Kořeny mohou být nalezeny dosazením. Jednoduše $\alpha^0 = 1$ není kořenem protože $f(1) = 1$, Nyní podle pravidel sčítání vyzkoušíme zda α^1 je kořen:

$$f(\alpha)=1+\alpha+\alpha^3=1+\alpha^0=0$$
rovnice 3.22

α^1 je tedy kořen, nyní vyzkoušíme zda α^2 je kořen:

$$f(\alpha^2)=1+\alpha^2+\alpha^6=1+\alpha^0=0$$
rovnice 3.23

α^2 je tedy také kořen, nyní vyzkoušíme zda α^3 je kořen:

$$f(\alpha^3)=1+\alpha^3+\alpha^9=1+\alpha^3+\alpha^2=1+\alpha^5=\alpha^4\neq 0$$
rovnice 3.24

α^3 tedy kořen není, je kořen α^4?

$$f(\alpha^3)=1+\alpha^4+\alpha^{12}=1+\alpha^4+\alpha^5=1+\alpha^0=0$$
rovnice 3.25

Ano α^4 je kořen. Nalezli jsme tedy všechny kořeny polynomu $f(X) = 1 + X + X^3$, jsou to α^1, α^2 a α^4. Jak je vidět všechny tři kořeny odpovídají některému ze sedmi nenulových elementů pole. Pravidlo, že primitivní polynom musí mít alespoň jeden kořen z primitivních elementů tento polynom bezezbytku splňuje.

3.6 Reed-Solomonovo kódování

Rovnice 3.2 připomenutá v rovnici 3.26 vyjadřuje nejběžnější formu Reed-Solomonových (R-S) kódů co se týče parametrů *n, k, t,* a jakéhokoli celého kladného čísla $m > 2$.

$$(n,k)=(2^m-1,2^m-1-2\mathrm{t})$$
rovnice 3.26

Kde $n - k = 2t$ je počet paritních symbolů a t počet symbolů, které má kód schopnost opravit. Generující polynom pro Reed-Solomonův kód má následující tvar:

$$g(X)=g_0+g_1X+g_2X^2+\ldots+g_{2\mathrm{t}-1}X^{2\mathrm{t}-1}+X^{2\mathrm{t}}$$
rovnice 3.27

Stupeň generujícího polynomu je stejný jako počet paritních symbolů. R-S kódy jsou podmnožinou Bose, Chaudhuri, and Hocquenghem (BCH) kódů, tedy právě odtud se vzal poměr mezi stupněm generujícího polynomu a počtem paritních symbolů, stejně jako u BCH kódů. Protože je generující polynom stupně $2t$, musí zde být přesně $2t$ mocnin α jdoucích po sobě, jež jsou kořeny polynomu. Jako kořeny polynomu $g(X)$ označíme α, α^2,…, α^{2t}. Není nutné začínat kořenem α, můžeme začít jakoukoli mocninou α chceme. Vezměme v úvahu symbolově orientovaný opravný R-S kód (7, 3). Generující polynom s $2t = n - k = 4$ kořeny lze popsat takto:

$$g(X)=(X-\alpha)(X-\alpha^2)(X-\alpha^3)(X-\alpha^4)$$
$$g(X)=(X^2-(\alpha+\alpha^2)X+\alpha^3)(X^2-(\alpha^3+\alpha^4)X+\alpha^7)$$
$$g(X)=(X^2-\alpha^4X+\alpha^3)(X^2-\alpha^6X+\alpha^0)$$
$$g(X)=X^4-(\alpha^4X+\alpha^6)X^3+(\alpha^3X+\alpha^{10}+\alpha^0)X^2-(\alpha^4+\alpha^9)X+\alpha^3$$
$$g(X)=X^4-\alpha^3X^3+\alpha^0X^2-\alpha^1X+\alpha^3$$
rovnice 3.28

Následuje přepis od nízkých řádů k vysokým a změna negativních znamének na pozitivní, protože v binárním poli +1 = -1, $g(X)$ pak bude vypadat:

$$g(X)=\alpha^3+\alpha^1X+\alpha^0X^2+\alpha^3X^3+X^4$$
rovnice 3.29

3.7 kódování v systematické formě

Protože jsou R-S kódy cyklické kódování v systematické formě je analogické k binárnímu kódování. Kódování můžeme brát jako posun polynomu zprávy, $m(X)$ vpravo o k kroků a poté připojit paritní polynom $p(X)$ zleva v $n - k$ krocích. Poté vynásobíme $m(X)$ a X^{n-k}, dále manipulací polynomu zprávy algebraicky tak, že ho posuneme vpravo o $n - k$ pozic. Dále vydělíme $X^{n-k}\,m(X)$ generujícím polynomem $g(X)$ což se dá zapsat následovně:

$$X^{n-k}m(X)=q(X)g(X)+p(X)$$
rovnice 3.30

Kde $q(X)$ a $p(X)$ jsou podíl a zbytek po dělení polynomu. Stejně jako v binárním případě zbytek po dělení je parita, rovnice 3.30 lze také vyjádřit takto:

$$p(X) = X^{n-k} m(X) \, modulo \, g(X)$$
rovnice 3.31

Výsledný polynom kódového slova $U(X)$ lze zapsat jako:

$$U(X) = p(X) + X^{n-k} m(X)$$
rovnice 3.32

V následujícím příkladu demonstrujeme kroky popsané rovnicemi 3.31 a 3.32 zakódováním následujícího zprávy o třech symbolech:

010 110 111
α^1 α^3 α^5

S R-S (7, 3) kde generující polynom je daný rovnicí 3.29. Nejdříve musíme vynásobit (posunout nahoru) polynom zprávy $\alpha^1 + \alpha^3 X + \alpha^5 X^2$ o X^{n-k}, což znemená o X^4, což vyjde $\alpha^1 X^4 + \alpha^3 X^5 + \alpha^5 X^6$, poté vydělíme posunutý polynom zprávy generujícím polynomem z rovnice 3.29. Dělení polynomů s nebinárními koeficienty není tak zřejmé jako u jeho binárního protějšku, protože jsou zde použity operace přičítání (odečítání) a násobení (dělení) které musí následovat pravidla pro dělení a násobení nad Galoisovým polem. Pro čtenáře je poté velmi obtížné jak toto dělení polynomů vyústí v následný zbytek po dělení (paritní) polynom.

$$p(X) = \alpha^0 + \alpha^2 X + \alpha^4 X^2 + \alpha^6 X^3$$

Poté z rovnice 3.32 polynom kódového slova můžeme napsat jako:

$$U(X) = \alpha^0 + \alpha^2 X + \alpha^4 X^2 + \alpha^6 X^3 + \alpha^1 X^4 + \alpha^3 X^5 + \alpha^5 X^6$$
rovnice 3.33

3.8 Reed-Solomonovo dekódování

Budeme pokračovat v příkladu zavedeném výše. Testovací zpráva zakódovaná v symetrické formě požitím R-S kódu (7, 3) nám vyšla jako polynom kódového slova popsaný v rovnici 3.33. Nyní, předpokládejme, že během přenosu se toto kódové slovo nějak poškodilo a dva ze symbolů byli přijaty chybně. Tento počet je mimo jiné maximum, co je tento R-S kód schopen opravit. Pro toto sedmi symbolové kó-

dové slovo označme jeho chybový polynom jako $e(X)$. V polynomiální formě vypadá následovně:

$$e(X)=\sum_{n=0}^{6} e_n X^n$$

rovnice 3.34

Nechť je pro tento příklad chyba dvou symbolů zapsána takto:

$$e(X)=0+0\mathrm{X}+0\mathrm{X}^2+\alpha^2 X^3+\alpha^5 X^4+0\mathrm{X}^5+0\mathrm{X}^6$$
$$e(X)=(000)+(000)X+(000)X^2+(001)X^3+(111)X^4+(000)X^5+(000)X^6$$

rovnice 3.35

Jinými slovy, jeden paritní symbol byl poškozen jedno bitovou chybou (vidíme ho jako α^2) a jeden datový symbol byl poškozen tří bitovou chybou (vidíme ho jako α^5). Přijatý polynom poškozeného kódového slova $r(X)$ je reprezentován jako součet polynomu přeneseného kódového slova a chybového polynomu:

$$r(X)=U(X)+e(X)$$

rovnice 3.36

Podle předešlé rovnice 3.36 sečteme rovnici kódového slova $U(X)$ (3.33) a rovnici chybového polynomu $e(X)$ (3.35), to nám dá $r(X)$:

$$r(X)=(100)+(001)X+(011)X^2+(100)X^3+(101)X^4+(110)X^5+(111)X^6$$
$$r(X)=\alpha^0+\alpha^2 X+\alpha^4 X^2+\alpha^0 X^3+\alpha^6 X^4+\alpha^3 X^5+\alpha^5 X^6$$

rovnice 3.37

V tomto příkladu máme čtyři neznámé, dvě chybná umístění a dvě chybné hodnoty. Připomeňme důležitý rozdíl mezi nebinárním dekódováním $r(X)$, který je vyjádřený v rovnici 3.37 a binární dekódováním. V binárním dekódování dekodér musí najít jen pozici chyby, hodnota je jasná, na místě chyby je jen třeba „překlopit“ bit z 1 do 0 nebo obráceně. Ovšem zde u nebinárních symbolů je třeba nejenom nalézt místo kde se chyba stala, ale také správnou původní hodnotu symbolu. Protože máme čtyři neznámé je pro jejich vyřešení třeba zavést čtyři rovnice.

3.9 Výpočet syndromů

Pojem „syndrom“ je výsledek kontroly parity nad přijatým kódovým slovem $r(X)$, který má za účel zjistit zda hodnota ze sestavy polynomu $r(X)$ je validní nebo ne. Pokud je hodnota validním členem její syndrom S je nulový. Jakákoli nenulová hodnota syndromu S znamená prezenci chyby. Podobně jako v binárním případě je syndrom S tvořen z n - k symbolů, $\{S_i\}$ (i = 1, … , n – k). Tedy pro R-S kód (7, 3) jsou v každém vektoru syndromu čtyři symboly. Jejich hodnoty se dají vypočítat z přijatého polynomu $r(X)$. Připomeňme si jak je kódové slovo počítáno, podle rovnice napsané níže:

$$U(X) = m(X)\,g(X)$$

rovnice 3.38

Z této rovnice je znatelné, že každé validní polynom kódového slova $U(X)$ je násobek generujícího polynomu $g(X)$. Následně kořeny $g(X)$ musí bý také kořeny $U(X)$. Jelikož se $r(X) = U(X) + e(X)$, tak $r(X)$ bude po dosazení kořenů $g(X)$ nulová jen pokud je $r(X)$ validní. Jakákoli chyba způsobí, že výsledek jednoho nebo více výpočtů bude nerovno nule. Výpočet symbolu syndromu lze vyjádřit takto:

$$S_i = r(X)_{X=a^i} = r(\alpha^i) \qquad i = 1, \ldots, n-k$$

rovnice 3.39

Když $r(X)$ obsahuje chybu ve dvou symbolech, jak je naznačeno v rovnici 3.37, všechny jeho syndromy S_i by neměli být rovné nule. Pro tento případ syndromy jsou nalezeny takto:

$$S_1 = r(\alpha) = \alpha^0 + \alpha^3 + \alpha^6 + \alpha^3 + \alpha^{10} + \alpha^8 + \alpha^{11}$$
$$S_1 = \alpha^0 + \alpha^3 + \alpha^6 + \alpha^3 + \alpha^2 + \alpha^1 + \alpha^4$$
$$S_1 = \alpha^3$$

rovnice 3.40

$$S_1 = r(\alpha^2) = \alpha^0 + \alpha^4 + \alpha^8 + \alpha^6 + \alpha^{14} + \alpha^{13} + \alpha^{17}$$
$$S_1 = \alpha^0 + \alpha^4 + \alpha^1 + \alpha^6 + \alpha^0 + \alpha^6 + \alpha^3$$
$$S_1 = \alpha^5$$

rovnice 3.41

$$S_1 = r(\alpha^3) = \alpha^0 + \alpha^5 + \alpha^{10} + \alpha^9 + \alpha^{18} + \alpha^{18} + \alpha^{23}$$
$$S_1 = \alpha^0 + \alpha^5 + \alpha^3 + \alpha^2 + \alpha^4 + \alpha^4 + \alpha^2$$
$$S_1 = \alpha^6$$

rovnice 3.42

$$S_1 = r(\alpha^4) = \alpha^0 + \alpha^6 + \alpha^{12} + \alpha^{12} + \alpha^{22} + \alpha^{23} + \alpha^{29}$$
$$S_1 = \alpha^0 + \alpha^6 + \alpha^5 + \alpha^5 + \alpha^1 + \alpha^2 + \alpha^1$$
$$S_1 = 0$$

rovnice 3.43

Výsledky potvrzují, že přijaté kódové slovo obsahuje chyby, čehož jsme chtěli dosáhnout, protože $S \neq 0$

Existuje další způsob jak vypočítat syndromy a to v případě, že známe chybový polynom $e(X)$. Poté se syndromy vypočítají opět dosazením kořenů $g(X)$ tentokráte do polynomu $e(X)$. výsledky syndromů S musí být totožné jako při dosazení kořenů do polynomu $r(X)$.

3.10 Lokalizace Chyb

Předpokládejme, máme v chyb v kódovém slově na pozicích X^{j_1}, X^{j_2}, … , X^{j_v}. Chybový polynom $e(X)$ z rovnic 3.34 a 3.35 můžeme poté psát jako:

$$e(X) = e_{j_1} X^{j_1} + e_{j_2} X^{j_2} + \ldots + e_{j_v} X^{j_v}$$

rovnice 3.44

Indexy 1, 2, … , v odkazují na první, druhou až v-tou chybu a j indexy odkazují na polohu chyby. Abychom mohli poškozené kódové slovo opravit, musí být nalezena každá z chybných hodnot e_{j_l} a jejich umístění X^{j_v} kde $l = 1, 2, \ldots, v$. Definujme chybové lokační číslo jako $\beta_l = \alpha^{j_l}$. Dále obdržíme $n - k = 2t$ syndromů tím, že substitucí vložíme α^i do přijatého polynomu pro $i = 1, 2, \ldots, 2t$:

$$S_1 = r(\alpha) = e_{j_1}\beta_1 + e_{j_2}\beta_2 + e_{j_v}\beta_v$$
$$S_2 = r(\alpha^2) = e_{j_1}\beta_1^2 + e_{j_2}\beta_2^2 + e_{j_v}\beta_v^2$$
$$\vdots$$
$$S_{2t} = r(\alpha^{2t}) = e_{j_1}\beta_1^{2t} + e_{j_2}\beta_2^{2t} + e_{j_v}\beta_v^{2t}$$

rovnice 3.45

Je zde 2*t* neznámých (*t* chybných hodnot a *t* pozic) a 2*t* rovnic. 2*t* rovnic najednou nejde vyřešit běžnou cestou protože jsou nelineární (některé neznámé mají exponenty). Jakákoli technika, která vyřeší takovýto systém rovnic se nazývá Reed-Solomonův dekódovací algoritmus.

Jakmile je vypočítán nenulový vektor syndromů (alespoň jeden z jeho symbolů je nenulový) znamená to, že byla přijata chyba. Je tedy nutné nalézt pozici chyby nebo chyb v přijatém slovu. Polynom lokalizace chyb $\sigma(X)$ lze vyjádřit takto:

$$\sigma(X)=(1+\beta_1 X)(1+\beta_2 X)\ldots(1+\beta_v X)$$
$$\sigma(X)=1+\sigma_1 X+\sigma_2 X^2+\ldots+\sigma_v X^v$$

rovnice 3.46

Kořeny $\sigma(X)$ jsou $1/\beta_1$, $1/\beta_2$, … , β_v. Převrácená hodnota kořenů $\sigma(X)$ jsou hodnoty lokalizačních chyb v chybovém polynomu $e(X)$. Použitím autoregresní modelující techniky ze syndromů zformujeme matici, kde prvních *t* syndromů je použito pro predikování dalšího syndromu.

$$\begin{bmatrix} S_1 & S_2 & S_3 & \ldots & S_{t-1} & S_t \\ S_2 & S_3 & S_4 & \ldots & S_t & S_{t+1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ S_{t-1} & S_t & S_{t+1} & \ldots & S_{2t-3} & S_{2t-2} \\ S_t & S_{t+1} & S_{t+2} & \ldots & S_{2t-2} & S_{2t-1} \end{bmatrix} \begin{bmatrix} \sigma_t \\ \sigma_{t-1} \\ \vdots \\ \sigma_2 \\ \sigma_1 \end{bmatrix} = \begin{bmatrix} -S_{t+1} \\ -S_{t+2} \\ \vdots \\ -S_{2t-1} \\ -S_{2t} \end{bmatrix}$$

rovnice 3.47

Nyní se opět vrátíme k našemu příkladu s R-S kódem (7, 3) a předešlé výsledky vypočítaných syndromů aplikujeme autoregresní technikou z rovnice 3.47 použitím matice o největší možné dimenzi, která má nenulový determinant. Pro dvou symbolový opravný R-S (7, 3) kód je to matice dimenze 2 × 2, model bude zapsaný takto:

$$\begin{bmatrix} S_1 & S_2 \\ S_2 & S_3 \end{bmatrix} \begin{bmatrix} \sigma_2 \\ \sigma_1 \end{bmatrix} = \begin{bmatrix} S_3 \\ S_4 \end{bmatrix}$$

rovnice 3.48

$$\begin{bmatrix} \alpha^3 & \alpha^5 \\ \alpha^5 & \alpha^6 \end{bmatrix} \begin{bmatrix} \sigma_2 \\ \sigma_1 \end{bmatrix} = \begin{bmatrix} \alpha^6 \\ 0 \end{bmatrix}$$

rovnice 3.49

K vyřešení koeficientů σ_1 *a* σ_2 chybového lokačního polynomu $\sigma(X)$ nejprve provedeme inverzi matice z rovnice 3.49. Inverze matice $[A]$ se provádí podle následujícího vzoru:

$$\mathrm{Inv}[A]=\frac{\mathrm{cofactor}[A]}{\det[A]}$$

rovnice 3.50

Poté tedy pokračování rovnice 3.50:

$$\det\begin{bmatrix}\alpha^3 & \alpha^5\\ \alpha^5 & \alpha^6\end{bmatrix}=\alpha^3\alpha^6-\alpha^5\alpha^5=\alpha^9-\alpha^{10}=\alpha^2-\alpha^3=\alpha^2+\alpha^3=\alpha^5$$

rovnice 3.51

$$\mathrm{cofactor}\begin{bmatrix}\alpha^3 & \alpha^5\\ \alpha^5 & \alpha^6\end{bmatrix}=\begin{bmatrix}\alpha^6 & \alpha^5\\ \alpha^5 & \alpha^3\end{bmatrix}$$

rovnice 3.52

Máme vypočítaný determinant i zrcadlenou matici (cofactor[17]), můžeme tedy dosadit do rovnice pro inverzní matici:

$$\mathrm{Inv}\begin{bmatrix}\alpha^3 & \alpha^5\\ \alpha^5 & \alpha^6\end{bmatrix}=\frac{\begin{bmatrix}\alpha^6 & \alpha^5\\ \alpha^5 & \alpha^3\end{bmatrix}}{\alpha^5}=\alpha^{-5}\begin{bmatrix}\alpha^6 & \alpha^5\\ \alpha^5 & \alpha^3\end{bmatrix}=\alpha^2\begin{bmatrix}\alpha^6 & \alpha^5\\ \alpha^5 & \alpha^3\end{bmatrix}=\begin{bmatrix}\alpha^8 & \alpha^7\\ \alpha^7 & \alpha^5\end{bmatrix}=\begin{bmatrix}\alpha^1 & \alpha^0\\ \alpha^0 & \alpha^5\end{bmatrix}$$

rovnice 3.53

Ověření správnosti vypočítaných hodnot

Pokud byla inverze provedena správně vynásobení původní matice její inverzní maticí musí dát jednotkovou matici:

$$\begin{bmatrix}\alpha^3 & \alpha^5\\ \alpha^5 & \alpha^6\end{bmatrix}\begin{bmatrix}\alpha^1 & \alpha^0\\ \alpha^0 & \alpha^5\end{bmatrix}=\begin{bmatrix}\alpha^4+\alpha^5 & \alpha^3+\alpha^{10}\\ \alpha^6+\alpha^6 & \alpha^5+\alpha^{11}\end{bmatrix}=\begin{bmatrix}\alpha^1 & \alpha^0\\ \alpha^0 & \alpha^1\end{bmatrix}$$

rovnice 3.54

Pokračujeme v rovnici 3.49 kde započneme hledání umístění chyb vyřešením koeficientů chybového lokačního polynomu $\sigma(X)$.

17 cofactor – pojem z lineární algebry, značí zrcadlenou matici, viz [32]

$$\begin{bmatrix}\sigma_2\\ \sigma_5\end{bmatrix}=\begin{bmatrix}\alpha^1 & \alpha^0\\ \alpha^0 & \alpha^5\end{bmatrix}\begin{bmatrix}\alpha^6\\ 0\end{bmatrix}=\begin{bmatrix}\alpha^7\\ \alpha^6\end{bmatrix}=\begin{bmatrix}\alpha^0\\ \alpha^6\end{bmatrix}$$

rovnice 3.55

Z rovnic 3.46 a 3.55 vyjádříme $\sigma(X)$.

$$\sigma(X)=\alpha^0+\sigma_1 X+\sigma_2 X^2=\alpha^0+\alpha^6 X+\alpha^0 X^2$$

rovnice 3.56

Kořeny $\sigma(X)$ odpovídají pozici chyby. Jakmile získáme kořeny, budeme vědět i pozice chyb. Obecně, kořeny $\sigma(X)$ mohou být jeden nebo více prvků pole. Kořeny nalezneme dosazením všech prvků pole do $\sigma(X)$ polynomu, tam kde se polynom rovná nule je i lokace chyby přijatého slova.

$$\begin{aligned}
&\sigma(\alpha^0)=\alpha^0+\alpha^6+\alpha^0=\alpha^6\neq 0\\
&\sigma(\alpha^1)=\alpha^0+\alpha^7+\alpha^2=\alpha^2\neq 0\\
&\sigma(\alpha^2)=\alpha^0+\alpha^8+\alpha^4=\alpha^6\neq 0\\
&\sigma(\alpha^3)=\alpha^0+\alpha^9+\alpha^6=0\Rightarrow \text{CHYBA}\\
&\sigma(\alpha^4)=\alpha^0+\alpha^{10}+\alpha^8=0\Rightarrow \text{CHYBA}\\
&\sigma(\alpha^5)=\alpha^0+\alpha^{11}+\alpha^{10}=\alpha^2\neq 0\\
&\sigma(\alpha^6)=\alpha^0+\alpha^{12}+\alpha^{12}=\alpha^0\neq 0
\end{aligned}$$

rovnice 3.57

Z rovnice 3.46 je patrné, že pozice chyb jsou převrácené hodnoty kořenů polynomu. Tedy $\sigma(\alpha^3) = 0$ naznačuje, že jeden z kořenů existuje v $1/\beta_l = \alpha^3$ a $1/\alpha^3 = \alpha^4$. Podobně pro druhý kořen, $\sigma(\alpha^4) = 0$ naznačuje, že druhý z kořenů existuje v $1/\beta_{l'} = \alpha^4$ a $1/\alpha^4 = \alpha^3$, kde l a l' odpovídají první, druhé až v té chybě. V tomto případě máme dvě chyby ve dvou symbolech, chybový polynom bude tedy následující:

$$e(X)=e_{j_1}X^{j_1}+e_{j_2}X^{j_2}$$

rovnice 3.58

Dvě chyby byly nalezeny na pozicích α^3 α^4. Uveďme, že indexace umístění chyby je zcela libovolná, pro tento případ můžeme označit $\beta_l = \alpha^{j_l}$ hodnoty jako $\beta_1 = \alpha^{j_1} = \alpha^4$ a $\beta_2 = \alpha^{j_2} = \alpha^4$.

3.11 Hodnota chyb

V předchozím textu byli chyby označeny jako e_{ji} kde index j odkazuje na pozici chyby a index l odkazuje na pořadí chyby. Protože hodnota chyby je svázána s konkrétním umístěním chyby můžeme označení zjednodušit z e_{j1} na pouze e_1. Je to příprava na konečné nalezení hodnoty chyb e_1 a e_2, které jsou přidružené k lokacím $\beta_1 = \alpha^3$ a $\beta_2 = \alpha^4$, na to můžeme použít jakoukoli ze čtyř rovnic syndromů. Z rovnice 3.45 vybereme S_1 a S_2:

$$S_1 = r(\alpha) = e_1\beta_1 + e_2\beta_2$$
$$S_2 = r(\alpha^2) = e_1\beta_1^2 + e_2\beta_2^2$$

rovnice 3.59

Tyto rovnice můžeme zapsat do matice takto:

$$\begin{bmatrix} \beta_1 & \beta_2 \\ \beta_1^2 & \beta_2^2 \end{bmatrix}\begin{bmatrix} e_1 \\ e_2 \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \end{bmatrix}$$

rovnice 3.60

$$\begin{bmatrix} \alpha^3 & \alpha^4 \\ \alpha^6 & \alpha^8 \end{bmatrix}\begin{bmatrix} e_1 \\ e_2 \end{bmatrix} = \begin{bmatrix} \alpha^3 \\ \alpha^5 \end{bmatrix}$$

rovnice 3.61

K vyřešení dvou chybových hodnot e_1 a e_2 v matici 3.61 použijeme invertování matice jako v předchozím případě při výpočtu pozic.

$$\mathrm{Inv}\begin{bmatrix} \alpha^3 & \alpha^4 \\ \alpha^6 & \alpha^1 \end{bmatrix} = \frac{\begin{bmatrix} \alpha^1 & \alpha^4 \\ \alpha^6 & \alpha^3 \end{bmatrix}}{\alpha^3\alpha^1 + \alpha^6\alpha^4} = \frac{\begin{bmatrix} \alpha^1 & \alpha^4 \\ \alpha^6 & \alpha^3 \end{bmatrix}}{\alpha^4 + \alpha^3} = \alpha^{-6}\begin{bmatrix} \alpha^1 & \alpha^4 \\ \alpha^6 & \alpha^3 \end{bmatrix} =$$
$$= \alpha^1\begin{bmatrix} \alpha^1 & \alpha^4 \\ \alpha^6 & \alpha^3 \end{bmatrix} = \begin{bmatrix} \alpha^2 & \alpha^5 \\ \alpha^7 & \alpha^4 \end{bmatrix} = \begin{bmatrix} \alpha^2 & \alpha^5 \\ \alpha^0 & \alpha^4 \end{bmatrix}$$

rovnice 3.62

Nyní máme inverzní matici, vyřešíme tedy matici 3.61:

$$\begin{bmatrix} e_1 \\ e_2 \end{bmatrix} = \begin{bmatrix} \alpha^2 & \alpha^5 \\ \alpha^0 & \alpha^4 \end{bmatrix}\begin{bmatrix} \alpha^3 \\ \alpha^5 \end{bmatrix} = \begin{bmatrix} \alpha^5 + \alpha^{10} \\ \alpha^3 + \alpha^9 \end{bmatrix} = \begin{bmatrix} \alpha^5 + \alpha^3 \\ \alpha^3 + \alpha^2 \end{bmatrix} = \begin{bmatrix} \alpha^2 \\ \alpha^5 \end{bmatrix}$$

rovnice 3.63

3.12 Oprava přijatého slova chybovým polynomem

Z rovnic 3.58 a 3.63 vytvoříme očekávaný chybový polynom:

$$\hat{e}(X)=e_1 X^{j_1}+e_2 X^{j_2}=\alpha^2 X^3+\alpha^5 X^4$$

rovnice 3.64

Předvedený algoritmus opravil přijatý polynom, výpočtem odhadu přeneseného kódového slova a nakonec podal dekódovanou zprávu:

$$\hat{U}(X)=r(X)+\hat{e}(X)=U(X)+e(X)+\hat{e}(X)$$

rovnice 3.65

$$r(X)=(100)+(001)X+(011)X^2+(100)X^3+(101)X^4+(110)X^5+(111)X^6$$
$$\hat{e}=(000)+(000)X+(000)X^2+(001)X^3+(111)X^4+(110)X^5+(111)X^6$$
$$\hat{U}=(100)+(001)X+(011)X^2+(101)X^3+(010)X^4+(110)X^5+(111)X^6$$
$$=\alpha^0+\alpha^2 X+\alpha^4 X^2+\alpha^6 X^3+\alpha^1 X^4+\alpha^3 X^5\alpha^5 X^6$$

rovnice 3.66

Protože symboly zprávy jsou $k = 3$ symboly napravo dekódovaná zpráva je:

$$\begin{array}{ccc} 010 & 110 & 111 \\ \alpha^1 & \alpha^3 & \alpha^5 \end{array}$$

Což je přesně ta zpráva, kterou jsme dříve pro tento příklad zakódovali.

3.13 Výpočetní náročnost algoritmů

Výpočetní náročnost jednotlivých částí Reed-Solomonova kódu se velmi odvíjí od délky zpracovávaného slova, tedy respektive od rozměru Galoisovy tabulky GF(2^m). Složitost roste obecně exponenciálně se zvyšujícím se m. Při vytváření Galoisovy tabulky je složitost vždy $O(n)$ kde n je 2^m. Při vytváření generujícího mnohočlenu je složitost daná délkou paritního polynomu, který má $2t$ symbolů. Složitost je tedy $O(n$ + suma $i)$ kde $i = 1, \ldots, n$ a n je $2t$. U enkodovani je složitost dosti podobná jako v předcházejícím případě, vychází z algoritmu lineárního posuvného zpětnovazebního registru, jehož složitost je $O(n$ + suma $i)$ kde $i = 1, \ldots, n$ a n je délka datové části

slova *k*. Dekodér je dozajista nejnáročnější část algoritmu, s vyjímkou kdy je přijaté slovo nepoškozeno, tedy když ani jeden syndrom není nulový. Dekódování se skládá z pěti hlavních operací, výpočet syndromu, výpočtu lokalizačního a evaluačního polynomu podle algoritmu Berlekamp-Massey, nalezení pozice chyb, což odpovídá výpočtu kořenů lokalizačního mnohočlenu, dále výpočet velikosti chyb podle Forneyho algoritmu a oprava chyb, což je sečtení původního přijatého slova s polynomem očekávaných chyb. Výpočet syndromu má složitost založenou na součinu délky paritního slova a délky celého přeneseného slova, tedy O(2*tn*) kde 2*t* je délka paritního slova a *n* je délka celého přijatého slova. Vytvoření lokalizačního polynomu se skládá z vytvoření matice ze syndromů o velikosti počet chyb ve slově × počet lokací chyb, což vždy dá čtvercovou matici, protože jedna chyba odpovídá právě jednomu umístění. K vyřešení koeficientů lokalizačního mnohočlenu potřebujeme matici inverzní k matici vytvořené ze syndromů, to zahrnuje výpočet determinantu matice a vytvoření zrcadlené matice. Když oboje máme, vynásobíme matice o rozměrech počet chyb × počet lokací chyb a 1 × počet lokací chyb. Násobení matic odpovídá složitosti O(n^3). Nalezení kořenů lokalizačního polynomu se sestává z dosazení všech hodnot z Galoisova pole do mnohočlenu a jeho výpočet, pokud se výsledek rovná nule byl kořen nalezen. Obecná složitost je tedy $O(n)$, kde *n* je rozměr Galoisovy tabulky GF(2^m) pro dosazení všech položek pole + výpočet polynomu, který se provádí součtem jeho jednotlivých koeficientů. Tato operace má složitost $O(n)$, kde *n* je počet koeficientů lokalizačního mnohočlenu. Vypočtení velikosti chyb je téměř totožné jako u výpočtu lokalizačního polynomu, jen se k zformování matice místo samotných syndromů použijí jakékoli z rovnic syndromů. Další postup je již totožný jako u lokalizačního mnohočlenu. Posledním krokem je oprava chyb, což je pouze součet přijatého slova a chybového polynomu, složitost $O(n)$, kde *n* je délka celého slova. Na mnoha místech algoritmu se také provádí převod hodnot z polynomiální formy do indexová a obráceně, složitost tohoto je vždy $O(n)$, kde *n* je délka datové struktury ve které se převádí.

3.14 Zhodnocení

V této kapitole jsme demonstrovali funkci silné skupiny nebinárních blokových kódů, jímž jsou R-S kódy: Zvláště vhodné k opravě náhodných chyb při přenosu

informace. Protože se efektivita kódu zvyšuje s přibývající délkou slova mají R-S kódy zvláště svůj půvab. Mohou být použity na dlouhé řetězce znaků s nižší výpočetní náročností než jiné kódy nad daty podobné délky. Je to proto, že logika dekodéru pracuje s celými symboly oproti bitově založeným algoritmům. Zvláště používané jsou R-S kódy s délkou symbolu 8 bitů tedy jedno bytové, ty mají sice větší komplexnost než bitové kódy, ale zase větší propustnost.

4 Praktická část

Programu je napsán programovacími jazyky c, c++ a pro grafické rozhraní jsou použity grafické knihovny QT4.2.3. Z QT prostředí je také použito několik dalších tříd, pro operace v hlavním programu, jako otevření/uložení souboru (QFile a QdataStream, QFileDialog), práce s řetězci (QString a QbyteArray) a práce s časem (QTime).

Program obsahuje tři třídy, „rs“ odvozenou od QmainWindow, která se stará o hlavní program, třídu „reedsolomon“, ve které je implementováno enkódování a dekódování podle Reed-Solomonova algoritmu a třídu „galois“, která obsahuje Galoisovo pole a generující mnohočlen, její obsah a obsah jejích proměnných je při každém spuštění programu stejný, Galoisovo pole se totiž generuje podle stále stejného nerozdělitelného polynomu. Kód implementace Galoisova pole, nenerujícího mnohočlenu, enkodéru a dekodéru si vzal inspiraci z R-S enkoderu/dekoderu od Simon Rockliff, University of Adelaide, http://www.eccpage.com/rs.c.

4.1 Popis atributů a metod tříd

Ilustrace 14: Diagram tříd

Následující obrázek popisující diagram tříd byl vytvořen ze zdrojových souborů v programu Enterprise Architekt Trial.

Třída reedsolomon, atributy:

- alpha_to: int[nn+1] – pole celých čísel, obsahuje všechny hodnoty Galoisova pole α^i $i = 0, \ldots, nn - 1$
- bb: int[nn-kk] – pole celých čísel, obsahuje paritu
- gg: int[nn-kk+1] – pole celých čísel, obsahuje generující polynom
- index_of: int[nn+1] – pole celých čísel, obsahuje indexy pole alpha_to, podle vzoru: když alpha_to[i] = x pak index[x] = i, pole se používá pro rychlejší operace nad Galoisovým polem, odpadá tím vyhledávání v něm.
- recd: int[nn] – pole celých čísel, jeho obsahem je přijaté kódové slovo, prováději se v něm operace s kódovým slovem

- rsErr: bool – příznak zda algoritmus skončil chybou či ne

Třída reedsolomon, metody:

- decode(int): int* – metoda dekódující přijatý polynom, argument je pole typu int s přijatým slovem, návratová hodnota je adresa začátku pole s opraveným přijatým slovem
- encode(int): int* – metoda enkódující data originálního souboru, argument je pole typu int s daty souboru, návratová hodnota je adresa začátku pole s paritním polynomem
- get_rsErr() – klasická metoda typu Get zpřístupňující hodnotu privátního abributu rsErr
- reedsolomon(void) – defaultní konstruktor
- reedsolomon(int, int, int) – parametrický konstruktor, v parametru přijímá pole alpha_to[], index_of[] a gg[]
- ~reedsolomon(void) – defaultní destruktor

Třída galois, atributy:

- alpha_to: int[nn+1] – pole celých čísel, obsahuje všechny hodnoty Galoisova pole α^i $i = 0, \ldots, nn - 1$
- gg: int[nn-kk+1] – pole celých čísel, obsahuje generující polynom
- index_of: int[nn+1] – pole celých čísel, obsahuje indexy pole alpha_to, podle vzoru: když alpha_to[i] = x pak index[x] = i, pole se používá pro rychlejší operace nad Galoisovým polem, odpadá tím vyhledávání v něm.
- pp: int[mm+1] – pole celých čísel, obsahujicí nerozložitelný polynom

Třída galois, metody:

- galois(void) – defaultní konstruktor
- galois(int) – parametrický konstruktor, argument je pole s nerozdělitelným polynomem
- ~galois(void) – defaultní destruktor

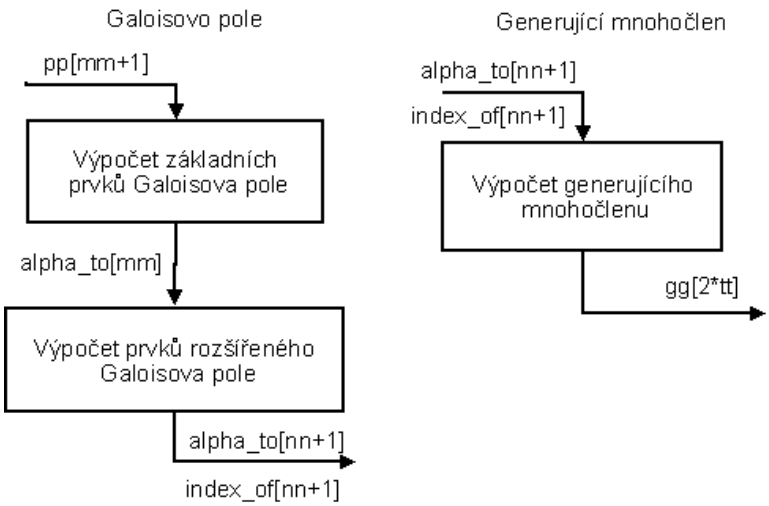
- generateGF(void) : void – medota generující Galoisovo pole, ukládá ho do alpha_to
- generateGP(void) : void – medota počítající generující mnohočlen, který uloží do gg
- get_alpha_to() – klasická metoda typu Get zpřístupňující privátního pole alpha_to
- get_gg() – klasická metoda typu Get zpřístupňující privátního pole gg
- get_alpha_to() – klasická metoda typu Get zpřístupňující privátního pole alpha_to

Třída rs, atributy

- alpha_to: int[nn+1] – pole celých čísel, obsahuje všechny hodnoty Galoisova pole α^i i = 0, …, nn - 1
- bb: int[nn-kk] – pole celých čísel, obsahuje paritu
- gf : galois – ukazatel na instanci třídy galois
- gg: int[nn-kk+1] – pole celých čísel, obsahuje generující polynom
- index_of: int[nn+1] – pole celých čísel, obsahuje indexy pole alpha_to, podle vzoru: když alpha_to[i] = x pak index[x] = i, pole se používá pro rychlejší operace nad Galoisovým polem, odpadá tím vyhledávání v něm.
- IrreduciblePolynomial: int[mm+1] – pole obsahující nerozložitelný polynom
- r_s : reedsolomon – ukazatel na instanci třídy reedsolomon
- test: int[nn] – pomocné pole použité pro testování výkonu algoritmu

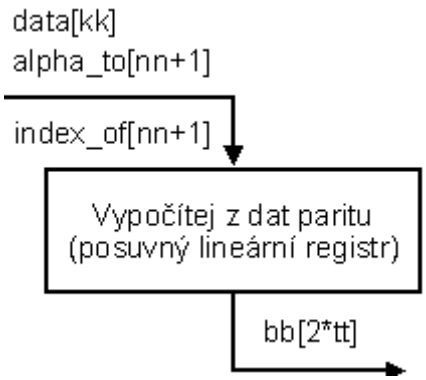
4.2 Blokové schéma programu

Následující schéma popisuje hlavní bloky implementované v programu, začneme částí programu generujícího Galoisovo pole a generující mnohočlen (15).



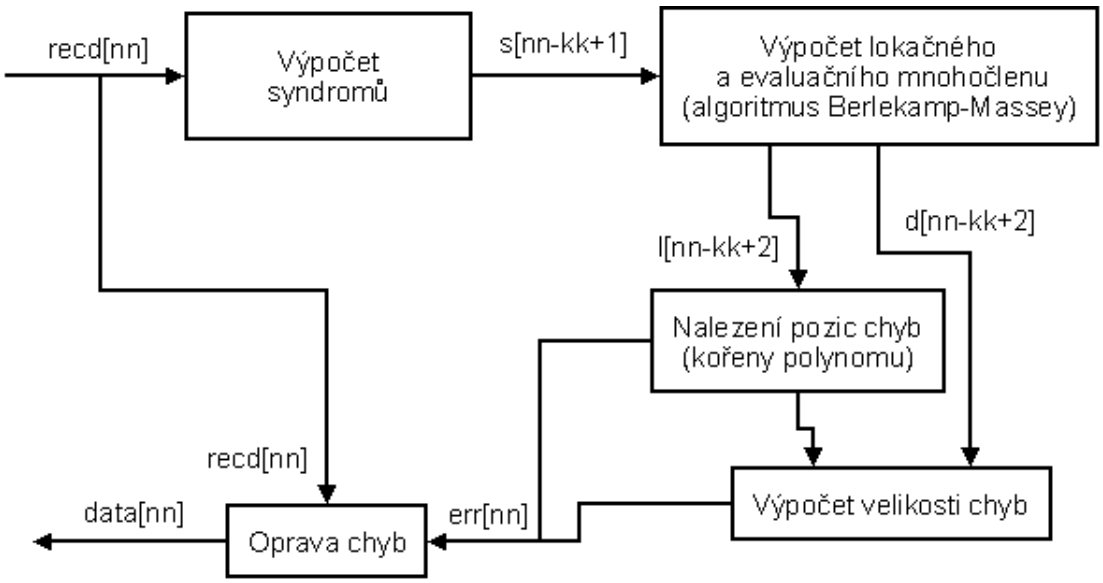
Ilustrace 15: schéma algoritmizace Galoisova pole a generijícího mnohočlenu

Pokračujeme v blokovém schématu enkodování (16):



Ilustrace 16: Reed-Solomonovo enkódování

Třetí schéma je dekódování (17):

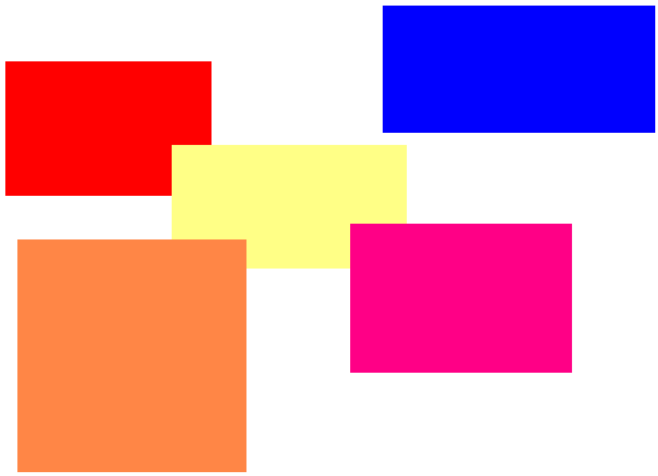


Ilustrace 17: Reed-Solomonovo dekódování

4.3 Ověření funkce programu

Pro ověření fukce programu, tedy to, že dokáže skutečně opravit datový soubor nějakým způsobem pozměněný oproti originálu, jsem pro ukázku zvolil bitmapový obrázek. Na bitmapě pozměněné nějakým překreslením v editoru obrázků je nejlépe vidět jak program dokáže data rekonstruovat. Z originálního souboru musel být vypočten paritní soubor, který se poté použije při pokusu o navrácení dat z poškozeného nebo změněného souboru.

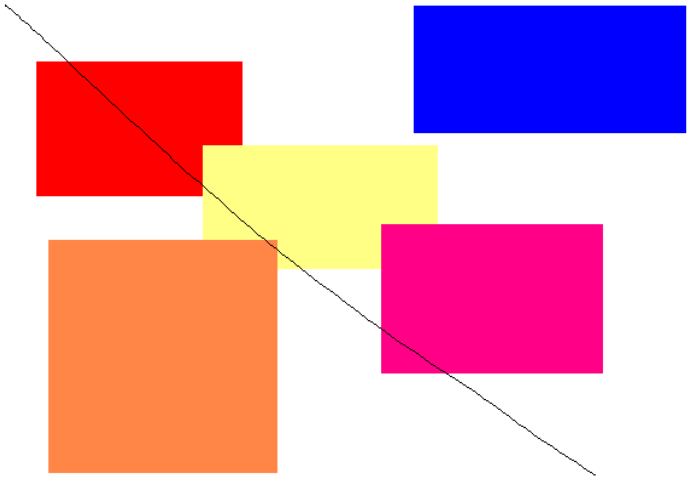
Originální obrázek (18) vypadá takto:



Ilustrace 18: Originální obrázek

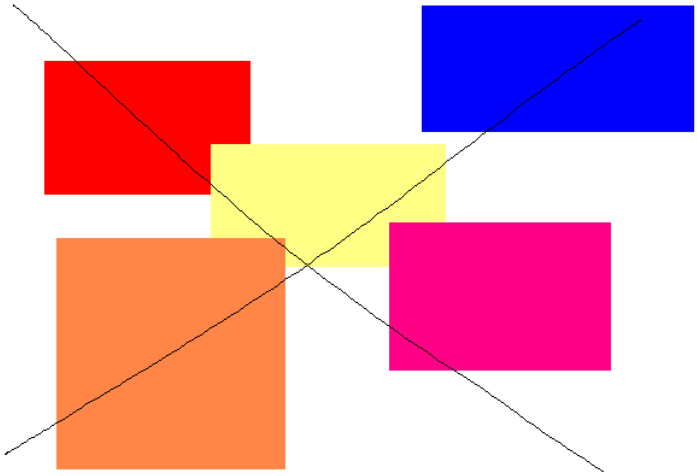
Obrázek má velikost 848 986 B, z něho byla vypočítána parita podle R-S kódu (255, 239), paritní byty tedy zabírají 16 B z celého kódového slova, což je přibližně 1/15 délky datové části slova, paritní soubor by tedy měl mít přibližně 1/15 velikosti souboru ze kterého byl vytvořen. Paritní soubor je velikosti 58 848 B.

Jako první příklad funkce programu uvedu pozměněný obrázek (19), který je algoritmus ještě schopen bezezbytku opravit:



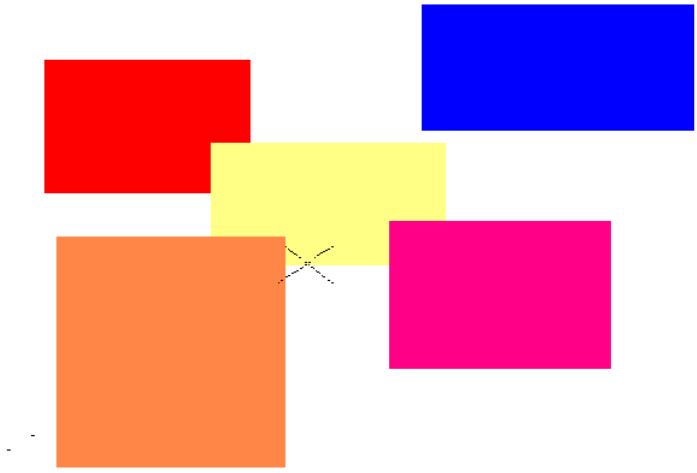
Ilustrace 19: Změněný obrázek opravitelný algoritmem

Výsledný obrázek vypadá stejně jako originál, všechny chyby na všech pozicích byli opraveny. Pokračujeme s obrázkem, který již plně opravitelný není (20):



Ilustrace 20: Lehce poškozený obrázek

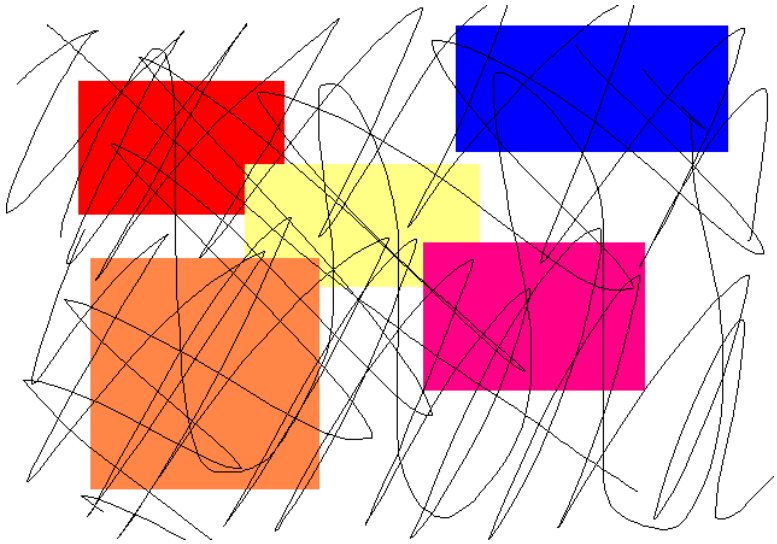
Výsledek vypadá takto (21):



Ilustrace 21: Výsledný obrázek s 21 chybami

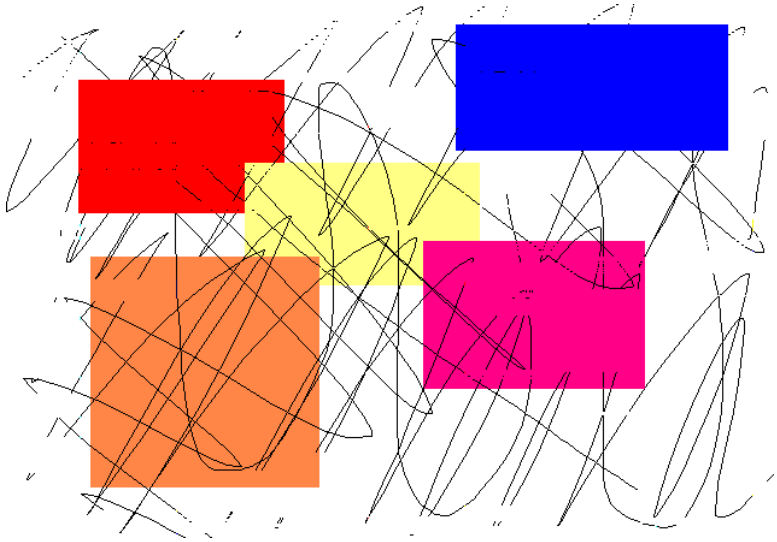
Jak je vidět problémové místo je uprostřed v místě křížení přikreslených čar, v tomto místě na mnoha pozicích dochází z překročení limitu osmi chybných bytů na jedno kódové slovo, což není nijak překvapující, když vezmeme v úvahu, že černá barva přikreslené čáry změní data souboru hned ve třech bytech, barva se sestává ze tří složek R-G-B, každá po jednom bytu.

Následuje nejtěžší případ, kdy je soubor tak poškozen, že R-S kód není schopen opravit téměř nic (22):



Ilustrace 22: Silně změněný obrázek

A výsledek silně poškozeného obrázku (23):



Ilustrace 23: Obrázek s 2136 chybami

Zde je změna tak masivní, že algoritmus neopravý téměř nic, jediná místa kde si vede ještě dobře je nad červeným (RGB FF0000) a modrým (RGB 0000FF) polem, protože zde černá (RGB 000000) linka změní jen dva ze tří bytů pixelu.

5 Závěr

Výsledkem mé bakalářské práce je popis nejpoužívanějšího způsobu zabezpečení dat proti jejich ztrátě za běhu a to RAID pole. Popis jeho základních levelů a způsobů implementace, jako je hardware a software řešení na dvou platformách, Windows a Linux. Dále teoretický popis Reed-Solomonových kódů s praktickými, názornými ukázkami funkce a způsobu výpočtu jednotlivých kroků algoritmu. Tyto ukázkové příklady byli většinou převzaté z materiálů od Berdarda Sklara [27], ale na některých místech mi připadlo vysvětlení nedostatečné, takže jsem ho ještě, snad vhodně, doplnil vlastním popisem problému a ukázkou na příkladu. Popis je natolik podrobný, že z něj lze tyto algoritmy implementovat v libovolném programovacím jazyce. Program vytvořený podle teorie popsané v práci se zdá býti funkční bez ohledu na typ dat, které zpracovává. Při programování jsem také narazil na několik nemilých bugů v samotných QT4 knihovnách a svůj program jsem musel tomuto přizpůsobit, aby byl stabilní. Jako hlavní problém spatřuji rozdílné chování aplikace mezi spuštěním ve vývojovém prostředí Visual Studia a spuštěním mimo něj, jen s potřebnými QT knihovnami. Někdy se chyby neprojevují ve vývojovém prostředí, ale až při spuštění mimo něj, je tak velmi těžké hledat problém. Samotná aplikace je velmi strohá, bylo by záhodno opatřit ji lepší grafickou prezentací i když se jedná pouze o demonstrační program. Do programu by se dále mohli zakomponovat algoritmy z BCH kódů, aby byl schopen opravit i výmazy z dat, toho samotný R-S kód není schopen. Program by měl být více ošetřen proti nekorektnímu zacházení s ním, soubory by měli být spřažené k sobě, aby bylo hned jasné zda konkrétní paritní soubor lze použít k opravě poškozeného originálního souboru, zda soubor má takouvou délku jakou má mít (R-S není schopen detekovat výmazy), použití hashů, změna délky parity (změna redundance) za běhu, apod. Svojí aplikací jsem se chtěl více přiblížit funkčnosti programu „dvdisaster“ http://dvdisaster.net/en/index.php, který mi byl původně jakýmsi vzorem, ale dotažení do takového stavu není pro jednoho člověka práce na týdny, ale na měsíce.

6 Zdroje

[1] Østergaard, J. Bueso, E. *The Software-RAID HOWTO* [online]. 2008 [cit. 2008-05-08]. Dostupný z WWW: http://www.tldp.org/HOWTO/Software-RAID-HOWTO.html

[2] Red Hat Linux 9. *Software RAID Configuration* [online]. 2008 [cit. 2008-05-08]. Dostupný z WWW: http://www.redhat.com/docs/manuals/linux/RHL-9-Manual/custom-guide/ch-software-raid.html

[3] WIKIPEDIA. *RAID* [online]. 2008 [cit. 2008-05-08]. Dostupný z WWW: http://en.wikipedia.org/wiki/RAID

[4] WIKIPEDIA. *RAID* [online]. 2008 [cit. 2008-05-08]. Dostupný z WWW: http://cs.wikipedia.org/wiki/RAID

[5] Čáslavka, D.. *Software RAID 1 ve Windows Server 2003 / XP* [online]. 2008 [cit. 2008-05-08]. Dostupný z WWW: http://ace-s.cz/modules.php?name=News&file=article&sid=130

[6] WIKIPEDIA. *Standard RAID levels* [online]. 2008 [cit. 2008-05-08]. Dostupný z WWW: http://en.wikipedia.org/wiki/Standard_RAID_levels

[7] WIKIPEDIA. *Non-standard RAID levels* [online]. 2008 [cit. 2008-05-08]. Dostupný z WWW: http://en.wikipedia.org/wiki/Non-standard_RAID_levels

[8] d3d. *Jak udělat ve windows XP software raid ?* [online]. 2008 [cit. 2008-05-08]. Dostupný z WWW: http://d3d.org/blog-kecy/jak-udelat-ve-windows-xp-software-raid

[9] WIKIPEDIA. *Disk array controller* [online]. 2008 [cit. 2008-05-08]. Dostupný z WWW: http://en.wikipedia.org/wiki/Disk_array_controller

[10] WIKIPEDIA. *RAID* [online]. 2008 [cit. 2008-05-08]. Dostupný z WWW: http://en.wikipedia.org/wiki/RAID

[11] WIKIPEDIA. *Mean time between failures* [online]. 2008 [cit. 2008-05-08]. Dostupný z WWW: http://en.wikipedia.org/wiki/MTBF

[12] WIKIPEDIA. *SCSI* [online]. 2008 [cit. 2008-05-08]. Dostupný z WWW: http://en.wikipedia.org/wiki/SCSI

[13] WIKIPEDIA. *Serial Storage Architecture* [online]. 2008 [cit. 2008-05-08]. Dostupný z WWW: http://en.wikipedia.org/wiki/-Serial_Storage_Architecture

[14] WIKIPEDIA. *Fibre Channel* [online]. 2008 [cit. 2008-05-08]. Dostupný z WWW: http://en.wikipedia.org/wiki/Fibre_Channel

[15] WIKIPEDIA. *Storage area network* [online]. 2008 [cit. 2008-05-08]. Dostupný z WWW: http://en.wikipedia.org/wiki/Storage_area_network

[16] WIKIPEDIA. *Logical block addressing* [online]. 2008 [cit. 2008-05-08]. Dostupný z WWW: http://en.wikipedia.org/wiki/Logical_block_addressing

[17] WIKIPEDIA. *Direct memory access* [online]. 2008 [cit. 2008-05-08]. Dostupný z WWW: http://en.wikipedia.org/wiki/Direct_memory_access

[18]WIKIPEDIA. *AT Attachment* [online]. 2008 [cit. 2008-05-08]. Dostupný z WWW: http://en.wikipedia.org/wiki/AT_Attachment

[19] WIKIPEDIA. *Peripheral Component Interconnect* [online]. 2008[cit. 2008-05-08]. Dostupný z WWW: http://en.wikipedia.org/wiki/-Peripheral_Component_Interconnect

[20] WIKIPEDIA. *Serial ATA* [online]. 2008[cit. 2008-05-08]. Dostupný z WWW: http://en.wikipedia.org/wiki/Serial_ATA

[21] WIKIPEDIA. *BIOS* [online]. 2008[cit. 2008-05-08]. Dostupný z WWW: http://en.wikipedia.org/wiki/BIOS

[22] WIKIPEDIA. *Plug-and-play* [online]. 2008[cit. 2008-05-08]. Dostupný z WWW: http://en.wikipedia.org/wiki/Plug_and_play

[23] WIKIPEDIA. *Self-Monitoring, Analysis, and Reporting Technology* [online]. 2008[cit. 2008-05-08]. Dostupný z WWW: http://en.wikipedia.org/wiki/Self-Monitoring%2C_Analysis%2C_and_Reporting_Technology

[24] WIKIPEDIA. *RAID 2* [online]. 2008[cit. 2008-05-08]. Dostupný z WWW: http://en.wikipedia.org/wiki/Standard_RAID_levels#RAID_2

[25] Wooley, S. *Manual for mdadm - man mdadm* [online]. 2008[cit. 2008-05-08]. Dostupný z WWW: http://swoolley.org/man.cgi/mdadm

[26] Patterson, D. Katz, R. and Gibson, G. *A Case For Redundant Arrays of Inexpensive Disks*, Univerzity California – Berkeley, CA, 1988

[27] Sklar, B. *Reed-Solomon Codes* [online]. [cit 2008-05-10]. Dostupný z WWW: http://www.facweb.iitkgp.ernet.in/~pallab/mob_com/art_sklar7_reed-solomon.pdf

[28] Wicker, S. B. Bhargava, V. K. *Reed Solomon codes and their applications*, IEEE Press, 1994, ISBN: 0-78035391-9

[29] Reed, I. S. Solomon, G. „*Polynomial Codes Over Certain Finite Fields*“, SIAM Journal of Applied Math., vol. 8, 1960, pp. 300-304.

[30] WIKIPEDIA. *Finite field* [online]. 2008[cit. 2008-05-08]. Dostupný z WWW: http://en.wikipedia.org/wiki/Galois_field

[31] Thelin, J. *Files, Directories and Streams* [online]. [cit 2008-05-10]. Dostupný z WWW: http://www.digitalfanatics.org/projects/qt_tutorial/-chapter08.html

[32] WIKIPEDIA. *Cofactor (linear algebra)* [online]. 2008[cit. 2008-05-08]. Dostupný z WWW: http://en.wikipedia.org/wiki/Cofactor_(linear_algebra)

7 Přílohy

Demonstrační program a všechny obrázky použité v práci jsou na přiloženém CD.

Seznam ilustrací

Seznam tabulek

Seznam zkratek

RAID (Redundant Array of Independent Disks) – vícenásobné diskové pole nezávislých disků)

JBOD (Just a Bunch Of Disks) – (jen svazek disků), D bývá někdy interpretováno jako Drives (jen svazek jednotek).

MTBF (Mean time between failures) – střední doba mezi selháními

ATA (Advanced Technology Attachment) – postarší rozhraní řadičů disků

PCI (Peripheral Component Interconnect) – rozhraní v současné době používané na základních deskách pro připojování rozšiřujících karet

SATA (Serial Advanced Technology Attachment) – hojně používané rozhraní řadičů disků

BIOS (Basic Input Output System) – základní vstupně výstupní systém, obsahuje ho každá základní deska

SSA (Serial Storage Architecture) – sériový transportní protokol používaný na serverech, vyvynutý v r. 1990 firmou IBM

FC (Fibre Channel) – sítová technologie o rychlosti 1-12 gigabitů primárně používaná pro síťová úložiště

SAN (Storage Area Network) – síťová architektura k připojování síťových diskových polí k počítačům takovým způsobem, že se diskové pole operačnímu systému jeví jako přímo připojené

LBA (Logical Block Addressing) – schéma jak specifikovat umístění bloku dat na počítačovém záznamovém mediu, typicky sekundární zařízení jako jsou pevné disky

PIO (Programmed Input/Output) – metoda pro přesun dat mezi hlavním procesorem a periferiemi, jako jsou síťové adaptéry nebo ATA disky

SMART (Self-Monitoring, Analysis, and Reporting Technology) – monitorovací systém pro počítačové pevné disky

R-S (Reed-Solomon) – dvojice tvůrců R-S kódů

GF (Galois Field) – Galoisovo konečné pole