

Implementace souborového systému JFFS2 do operačního systému Freescale MQX

Implementation of JFFS2 file system into Freescale MQX
operating system

Bc. Jan Králík



Univerzita Tomáše Bati ve Zlíně

Fakulta aplikované informatiky

akademický rok: 2010/2011

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Jan KRÁLÍK**

Osobní číslo: **A09479**

Studijní program: **N 3902 Inženýrská informatika**

Studijní obor: **Informační technologie**

Téma práce: **Implementace souborového systému JFFS2 do
operačního systému Freescale MQX**

Zásady pro vypracování:

1. Seznamte se se souborovým systémem JFFS2, operačním systémem MQX firmy Freescale a tvorbou aplikací v tomto systému.
2. Zpracujte literární rešerši na téma JFFS2, možností jeho využití a implementace.
3. Navrhněte způsob implementace JFFS2 do OS MQX.
4. Navrženou implementaci realizujte – vytvořte programové vybavení umožňující využití JFFS2 v systému MQX.
5. Vytvořte ukázkovou aplikaci demonstrující použití vytvořeného programového vybavení.

Rozsah diplomové práce:

Rozsah příloh:

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

Seznam odborné literatury:

1. MANN, Burkhard. C pro mikrokontroléry. Praha : BEN – technická literatura, 2004. 280 s. ISBN 80-7300-077-6.
2. PINKER, Jiří. Mikroprocesory a Mikropočítače. Praha : BEN – technická literatura, 2004. 220 s. ISBN 80-7300-110-1.
3. CATSOULIS, John. Designing Embedded Hardware. O'Reilly Media, 2005. 400 s. ISBN 978-0-596-00755-3.
4. MORTON, Todd D. Embedded Microcontrollers, Prentice Hall, 2001. 694 s. ISBN 0-13-907577.
5. WOODHOUSE, David. JFFS : The Journalling Flash File System [online]. Red Hat, Inc., 2005 [cit. 2011-01-21]. Dostupný z WWW: [http://sourceware.org/jffs2/jffs2.pdf].
6. Freescale MQX? Software Solutions [online]. Freescale Semiconductor, Inc., 2011 [cit. 2011-01-21]. Dostupný z WWW: [http://www.freescale.com/webapp/sps/site/homepage.jsp?code=MQX.HOME].
7. Freescale MQX? Real-Time Operating System (RTOS) [online]. Freescale Semiconductor, Inc., 2011 [cit. 2011-01-21]. Dostupný z WWW: [http://www.freescale.com/webapp/sps/site/overview.jsp?code=MQXRTOS&tid=m32MQX].

Vedoucí diplomové práce:

Ing. Jan Dolinay, Ph.D.

Ústav automatizace a řídicí techniky

Datum zadání diplomové práce:

24. února 2011

Termín odevzdání diplomové práce:

18. května 2011

Ve Zlíně dne 24. února 2011

prof. Ing. Vladimír Vašek, CSc.
děkan



doc. Mgr. Roman Jašek, Ph.D.
ředitel ústavu

ABSTRAKT

Tato diplomová práce se zabývá implementací souborového systému JFFS2 do operačního systému Freescale MQX. V teoretické části jsou uvedeny informace o flash pamětech, dále je popsán souborový systém JFFS2. Také jsou vypsány základní informace o operačním systému Freescale MQX. V praktické části je nejdříve navržen způsob komunikace flash paměti a aplikací s ovladačem JFFS2. Následně je uveden návrh implementace souborového systému, popis vlastní realizace ovladače a výsledky testování implementovaných funkcí.

Klíčová slova: JFFS2, flash paměť, Freescale MQX

ABSTRACT

This thesis deals with implementation of JFFS2 file system into Freescale MQX operating system. In the theoretical part basic information about flash memories is presented. Then the JFFS2 file system is described and also the MQX operating system.

In the practical part the scheme of communication of flash memory with application and JFFS2 driver is described first. Then the design and implementation of the file system is explained, followed by the results of testing of the implemented functions.

Keywords: JFFS2, flash memory, Freescale MQX

Rád bych zde poděkoval panu Ing. Janu Dolinayovi, Ph.D. za cenné rady, podnětné připomínky a vedení při tvorbě této diplomové práce.

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně

.....
podpis diplomanta

OBSAH

ÚVOD.....	10
I TEORETICKÁ ČÁST	11
1 ÚVOD K FLASH PAMĚTÍM	12
1.1 FLASH PAMĚŤ	12
1.2 FLASH TRANSLATION LAYER (FTL)	13
1.3 SOUBOROVÉ SYSTÉMY PRACUJÍCÍ S FLASH PAMĚTI	13
2 ÚVOD K UNIXOVÝM SOUBOROVÝM SYSTÉMŮM.....	14
2.1 I-UZLY	14
2.2 ADRESÁŘE.....	14
3 JFFS2.....	15
3.1 ZPŮSOB ULOŽENÍ ZÁZNAMŮ A SEZNAM BLOKŮ	15
3.2 TYPY UZLŮ	16
3.2.1 JFFS2 NODETYPE INODE	16
3.2.2 JFFS2 NODETYPE DIRENT	17
3.2.3 JFFS2 NODETYPE CLEANMARKER	17
3.3 PROVOZ.....	17
3.4 PŘIPOJOVÁNÍ	19
3.5 GARBAGE COLLECTOR.....	20
4 FREESCALE MQX	22
4.1 ZÁKLADNÍ INFORMACE	22
4.2 OVLADAČ FLASHX.....	22
4.2.1 FLASHX_BLOCK_INFO_STRUCT	22
4.2.2 Další operace	22
II PRAKTICKÁ ČÁST	23
5 NÁVRH IMPLEMENTACE.....	24
5.1 SCHÉMA IMPLEMENTACE OVLADAČE JFFS2.....	24
5.2 FUNKCE NA ROZHRANÍ JFFS2 A FLASHX	25
5.3 FUNKCE NA ROZHRANÍ JFFS2 A APLIKACE	25
5.4 SIMULAČNÍ PROSTŘEDÍ	26
5.5 FYZICKÉ ULOŽENÍ NA DISKU	26
5.6 PŘIPOJOVÁNÍ MÉDIA (MOUNT)	26
5.7 PRÁCE S ADRESÁŘI.....	27
5.7.1 Výpis obsahu adresáře.....	27
5.7.2 Odstranění adresáře	27
5.7.3 Vytvoření adresáře	27

5.7.4	Změna aktuálního adresáře.....	28
5.8	PRÁCE SE SOUBORY	28
5.8.1	Vytvoření souboru	28
5.8.2	Otevření souboru	28
5.8.3	Uzavření souboru	28
5.8.4	Zápis do souboru	29
5.8.5	Čtení ze souboru.....	29
5.8.6	Odstranění souboru	29
5.9	ODPOJOVÁNÍ MÉDIA (UNMOUNT)	29
5.10	INICIALIZACE SOUBOROVÉHO SYSTÉMU NA MÉDIU	29
6	VLASTNÍ REALIZACE	30
6.1	FYZICKÉ ULOŽENÍ NA DISKU	30
6.2	PŘIPOJOVÁNÍ (MOUNT)	32
6.3	OPERACE S UZLY TYPU DIRENT.....	37
6.3.1	Odstranění dirent	38
6.3.2	Získání dirent dle cesty.....	38
6.3.3	Získání dirent dle názvu	38
6.3.4	Získání dirent rodičovského adresáře.....	41
6.4	OPERACE S UZLY TYPU INODE	43
6.4.1	Získání seznamu uzlů inode	43
6.4.2	Získání inode s nejvyšším číslem ino.....	43
6.4.3	Získání velikosti souboru	43
6.4.4	Načtení vlastních dat z inode	43
6.5	FUNKCE PRO ZÁPIS FYZICKÝCH UZLŮ	44
6.5.1	Zápis fyzického uzlu raw_dirent	44
6.5.2	Zápis fyzického uzlu raw_inode.....	45
6.5.3	Zápis logického uzlu dirent	45
6.5.4	Zápis logického uzlu inode	45
6.5.5	Získání pozice pro zápis dat.....	45
6.6	OPERACE S ADRESÁŘI.....	46
6.6.1	Změna adresáře.....	46
6.6.2	Odstranění adresáře	46
6.6.3	Vytvoření adresáře	46
6.6.4	Výpis obsahu adresáře.....	47
6.7	OPERACE SE SOUBORY	47
6.7.1	Otevření souboru	47
6.7.2	Zavření souboru.....	48
6.7.3	Zápis do souboru	48
6.7.4	Čtení ze souboru.....	48
6.7.5	Vytvoření souboru	49
6.7.6	Odstranění souboru	49

6.8	ODPOJOVÁNÍ (UNMOUNT)	49
6.9	INICIALIZACE ZAŘÍZENÍ.....	49
7	PŘEHLED SOUBORŮ OVLADAČE	50
8	TESTOVÁNÍ	51
8.1	INICIALIZACE SOUBOROVÉHO SYSTÉMU PRO MÉDIUM.....	51
8.2	VYTVOŘENÍ ADRESÁŘE NEBO SOUBORU	52
8.3	ZÁPIS DO SOUBORU	54
8.4	MAZÁNÍ ADRESÁŘE NEBO SOUBORU	55
9	UKÁZKOVÁ APLIKACE.....	57
9.1	NÁVOD KE SPUŠTĚNÍ APLIKACE NA PC	58
9.2	NÁVOD KE SPUŠTĚNÍ APLIKACE NA VÝVOJOVÉ DESCE	58
10	SHRNUTÍ.....	61
10.1	PŘIPOJOVÁNÍ	61
10.2	OBSAH SOUBORU, ADRESÁŘE	61
10.3	ZÁPIS.....	62
10.4	NÁVRHY PRO POSTUP DALŠÍ IMPLEMENTACE	62
ZÁVĚR		63
ZÁVĚR V ANGLIČTINĚ.....		64
SEZNAM POUŽITÉ LITERATURY		65
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK		66
SEZNAM OBRÁZKŮ		67
SEZNAM PŘÍLOH.....		68

ÚVOD

Souborový systém JFFS2 je určen výhradně pro využití na flash pamětech. Flash paměti jsou nenahraditelné především v oblasti konstrukce embedded zařízení a v počítačích pro řízení technologických procesů. Velkým problémem flash pamětí je jejich životnost, především při opakovaném zápisu na jedno místo. Je totiž dán maximální počet přepisů části flash paměti označované pojmem blok. Existuje několik způsobů řešení tohoto problému.

Jedním z nich je Flash Translation Layer (FTL), který bývá do flash pamětí zabudován jako řadič. Proto mohou být na flash zařízeních (např. USB flash disky) použity i souborové systémy, které jsou využívány na pevných discích (např. FAT 32).

FTL se tedy stará o rovnoměrné využití bloků flash i při opakovaném zápisu na jedno místo a je často využíván v paměťových kartách fotoaparátů, USB flash discích a dalších spotřebních zařízeních.

Dalším způsobem jak rozumně využívat životnost bloků je použití souborového systému určeného primárně pro práci pamětmi typu flash. Tyto souborové systémy řeší problém opakovaného zápisu vlastním postupem.

Může se jednat o systémy vyvinuté společnostmi pro jejich vlastní užití. Tedy o systémy specializované pro konkrétní druh zařízení. Příkladem je systém Research-In-Motion File System, který využívá společnost Research-In-Motion pro chytré mobilní zařízení.

Také existují souborové systémy pro širší využití. Jedním z nich je souborový systém JFFS2. Jeho předchůdce JFFS1 byl vytvořen společností Axis Communications. Systém JFFS2 vyvíjela společnost Redhat. Tento souborový systém je implementován v operačních systémech Linux a eCos. Právě implementací souborového systému JFFS2 do operačního systému Freescale MQX se zabývá tato diplomová práce.

I. TEORETICKÁ ČÁST

1 ÚVOD K FLASH PAMĚTÍM

Flash paměť je velmi často používaná pro ukládání dat v embedded zařízeních. Je to pevné úložiště s vysokou spolehlivostí, velkou kapacitou a relativně nízkou cenou.

1.1 Flash paměť

Následující popis vychází ze zdroje [1] a [2].

Flash paměť je ve své podstatě kombinace RAM a pevného disku ve formě „pevné karty“. Flash paměť uchovává elektronická data v paměťových buňkách stejně jako DRAM a SRAM, ale současně pracuje jako pevný disk, jelikož si uložené informace zachová i po odpojení od elektrického napájení.[2]

Flash paměť může být rozdělena na více částí o různé velikosti. Každá část je dále rozdělena na paměťové buňky o stejných velikostech.

Na flash pamětech je tedy možné trvale uchovávat zapsaná data. Výhodou je možnost nasazení do provozu, kde nelze např. kvůli prašnosti nebo rezonancím použít klasický pevný disk. Také je výhoda uspořené místa, protože flash paměti zabírají menší prostor než pevný disk.

Velkou nevýhodou je ovšem, že paměťové buňky mají omezenou životnost přibližně na 100 000 cyklů zápisu. S tím je spojený problém zápisu dat. Při opakovaném zápisu dat na začátku flash paměti, by mohlo docházet k opotřebení jedné či několika paměťových buněk, zatímco ostatní by zůstaly neopotřebené (toto chování je označováno jako wear-leveling).

Taktéž je nevýhodou, že každá paměťová buňka musí být smazána celá, nelze tedy smazat pouze její část.

Podle způsobu uspořádání elektronických obvodů dělíme flash paměti na typ NAND a NOR.

Velikost paměťové buňky pro typ NOR je typicky 128 KiB a pro typ NAND je 8 KiB. Dalším rozdílem je, že NAND bloky jsou skládány do stránek, typická velikost je 512 byte. Každá stránka obsahuje speciálních 16 byte pro metadata a kódy pro opravu chyb.

1.2 Flash translation layer (FTL)

Flash translation layer slouží k tomu, aby i souborové systémy určené primárně pro zápis na disk (např. FAT 32, NTFS) mohly využívat flash paměti, aniž by došlo k rychlému opotřebení těchto pamětí. FTL zaručuje delší životnost flash i při opakovaném zápisu na totéž místo. U většiny přenosových médií je FTL již interně zabudována jako řadič. FTL tedy využívají např. USB flash disky, nebo v poslední době rozmáhající se SSD disky, které právě využívají flash paměti a nahrazují klasické pevné disky.

1.3 Souborové systémy pracující s flash pamětmi

Také existují souborové systémy, které nevyužívají FTL a řeší využití životnosti flash paměti vlastním způsobem.

Kromě níže popisovaného systému JFFS2 mezi tyto souborové systémy patří:

- Research-In-Motion File System

Souborový systém společnosti Research In Motion, který je využit pro chytré mobilní telefony a další zařízení této společnosti.

- FFS2 (Flash File System)

Souborový systém navržený společností Microsoft, dnes nevyužívaný.

- TrueFFS (True Flash File System)

Systém vyvinutý firmou M-Systems, využívaný pro SSD disky.

- JFFS1

Předchůdce systému JFFS2. Byl vyvinut firmou Axis Communications.

- YAFFS (Yet Another Flash File System)

Souborový systém navržený pro paměti typu NAND

Dále existuje mnoho dalších souborových systémů využívaných pro konkrétní hardware.

Informace pocházejí ze zdroje [4]

2 ÚVOD K UNIXOVÝM SOUBOROVÝM SYSTÉMŮM

JFFS2 vychází z UNIXových souborových systémů, proto budou nastíněny základní údaje, především ty, které JFFS2 využívá.

2.1 I-uzly

Ve všech UNIXových souborových systémech se v různých podobách vyskytuje datová struktura nazývaná i-uzel (identifikační uzel, i-node). Tato struktura obsahuje metadata o jednom konkrétním souboru a je tedy jakousi "hlavičkou" souboru. I-uzel je identifikován svým číslem, které je v rámci jednoho svazku jednoznačné. V i-uzlu jsou uložena přístupová práva souboru, vlastník a skupina souboru, typ souboru (běžný soubor, adresář, roura a další), časy (modifikace souboru, přístupu k souboru a modifikace i-uzlu), počet odkazů, délka souboru, a také odkaz na datové bloky souboru. [3]

2.2 Adresáře

Adresář (directory) je v UNIXu v podstatě běžný soubor, jen má u sebe příznak, že jde o adresář. Obsah tohoto souboru je interpretován jako seznam dvojic: pro každý soubor je zde jméno souboru a číslo i-uzlu, který je pod tímto jménem dostupný. Takto zejména může být jeden soubor (i-uzel) dostupný pod více jmény, případně z více adresářů na tomtéž svazku (tzv. pevný odkaz, hard link).

Každý adresář obsahuje položku "." (odkaz sám na sebe) a položku ".." – odkaz na nadřazený adresář (v kořeni svazku odkaz sám na sebe).

Jméno souboru se v UNIXu interpretuje s rozlišováním velikosti písmen a může obsahovat libovolné znaky kromě lomítka (používá se pro oddělení komponent jména v rámci cesty) a nulového bajtu (používá se pro ukončení řetězců v jazyce C).[3]

3 JFFS2

Následující popis vychází ze zdroje [1].

Jak bylo uvedeno výše, JFFS2 je souborový systém navržený pro flash paměti. Odstraňuje chyby a nedostatky předchozích verzí JFFS a JFFS1. Především se mění filosofie garbage collectoru. JFFS byl vyvinut firmou Axis Communnations. JFFS2 byl vyvíjen firmou Redhat.

3.1 Způsob uložení záznamů a seznam bloků

JFFS2, na rozdíl od JFFS1, nepoužívá cyklický zápis dat. Ke každému bloku se přistupuje individuálně. Uzly nemohou překročit hranici vymazaného bloku tak jako u JFFS.

Uzly jsou ukládány sekvenčně za sebe, obsahují následující položky:

- Identifikaci souboru, pod který uzly patří.
- Číslo verze, které určuje aktuálnost dat v uzlech.
- Pozice dat v souboru.
- Vlastní data.

Ukázka zápisu na médium:

Médium	Akce uživatele
verze: 1 délka: 200 offset: 0 data: AAAA...	Zápis 200 byte 'A' na offset 0 do daného souboru
verze: 2 délka: 200 offset: 200 data: BBBB...	Zápis 200 byte 'B' na offset 200 do daného souboru
verze: 3 délka: 50 offset: 175 data: CCCC...	Zápis 50 byte 'C' na offset 175 do daného souboru

Obr. 1 Fyzický zápis na médium

Při zpětné rekonstrukci obsahuje soubor v rozsahu 0-175 písmena A, v rozsahu 175-225 písmena C a v rozsahu 225-400 písmena B.

Uzly jsou označeny jako zastaralé, pokud jsou později zapsána data na stejnou pozici v souboru. A tvoří „špinavé místo“ v souborovém systému.

V ukázkovém případě by proběhl další zápis v rozsahu 0-175 například písmeny D. Pak jsou původní písmena A kompletně přepsána a uzel je neplatný.

3.2 Typy uzlů

JFFS zahrnoval pouze 1 typ uzlů, JFFS2 používá více typů. Každý uzel začíná hlavičkou, která obsahuje délku, typ uzlu a CRC. Toto je hlavička společná pro všechny druhy uzlů. Každý typ uzlu pak obsahuje další data, podle potřeby daného uzlu.

MSB		LSB	
Magic Bitmask 0x19 0x85		Typ uzlu	
Velikost uzlu			
CRC hlavičky			

Obr. 2 Hlavička společná pro všechny uzly

3.2.1 JFFS2 NODETYPE INODE

Tento typ uzlu reprezentuje soubor nebo adresář. Obsahuje metadata a také rozsah dat spadajících do uzlu. Nicméně neobsahuje název souboru či adresáře nebo počet mateřských uzlů. Pokud se jedná o soubor, obsahuje i samotná data. Pokud se jedná o adresář, neobsahuje žádná další data

Data spojená s tímto uzlem mohou být komprimována mnoha kompresními algoritmy, připojenými k JFFS2. Nejjednodušší je „žádný“, to znamená, že data nejsou komprimována. Dalším druhem je ZERO, kdy jdou data samé nuly a je udána jejich pozice a velikost.

3.2.2 JFFS2 NODETYPE DIRENT

Tento uzel reprezentuje záznam o souboru či adresáři. Obsahuje číslo uzlu rodičovského adresáře, ve kterém se záznam nachází. Dále název souboru či adresáře a číslo uzlu inode, na který záznam ukazuje.

Smazání adresáře se provádí tak, že se vytvoří uzel DIRENT se stejným jménem, ale patřící k uzlu 0. Je nutné, aby číslo verze nového uzlu bylo vyšší, než číslo verze předchozího uzlu.

3.2.3 JFFS2 NODETYPE CLEANMARKER

Tento uzel je zapsán do čerstvě smazaného bloku a určuje, že operace smazání byla provedena úspěšně a že je možné zapsat data.

Původní JFFS jednoduše předpokládal, že pokud má každý byte v bloku hodnotu 0xFF (tedy obsahuje samé jedničky) je blok volný. Ale při testování se ukázalo, že toto řešení nebylo nejšťastnější. Pokud došlo k odpojení napájení během procesu mazání, tak potom u mnoha flash čipů došlo k tomu, že některé bity zůstaly v nestálém stavu. Následně bylo špatně vyhodnoceno, zda-li je blok prázdný a došlo ke ztrátě dat.

Pokud JFFS2 narazí na bloky, které nemají na začátku požadovanou hlavičku, spustí následně operaci mazání a vloží uzel CLEANMARKER.

3.3 Provoz

Zápis probíhá podobně jako u JFFS, uzly jsou zapisovány sekvenčně, dokud není blok zaplněn. Nový blok je vybrán z *free_list* a zápis pokračuje od začátku nového bloku.

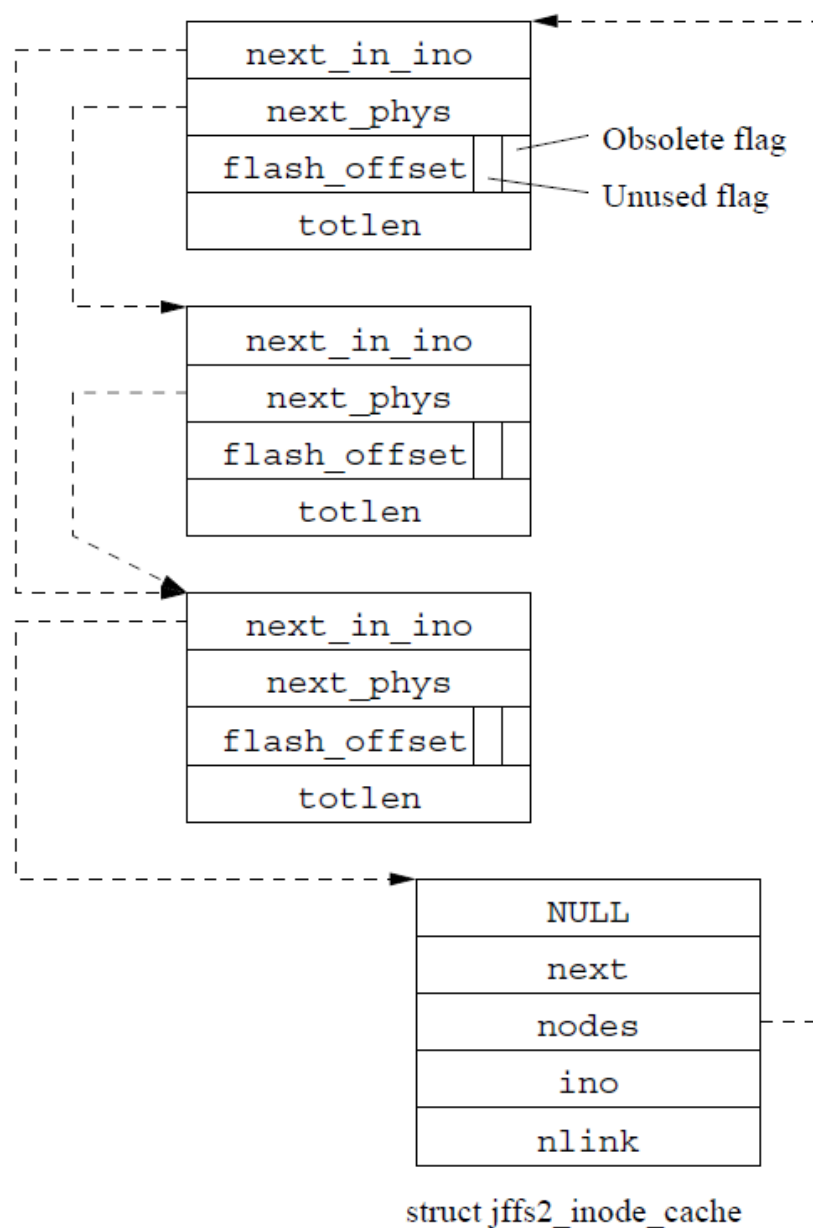
Pokud je *free_list* vyčerpán, pak se spustí garbage collector, který přesouvá uzly ze staršího bloku do nového bloku tak dlouho, dokud lze mazat starší bloky.

JFFS2 si po celou dobu nepamatuje všechny informace o uzlech. Během připojování média je sestavena celá mapa uzlů, ale struktury udržované v paměti jsou omezeny na velikost, bez které nelze mapu sestavit rychle na požádání.

Pro každý soubor či adresář existuje *struct jffs2_inode_cache*. Tato struktura obsahuje ukazatel na další inode cache v řádku hash tabulky. Dále obsahuje ukazatel na začátek dynamického seznamu fyzických uzlů, které patří pod tento soubor či adresář. Také je

uloženo identifikační číslo souboru či adresáře a počet odkazů na tuto inode cache. Tyto struktury jsou uloženy v hash tabulce. V každé buňce je dynamický seznam. Hashovací funkce je velmi jednoduchá – pouze číslo i-uzlu modulo velikost hash tabulky.

Každý fyzický uzel je reprezentován strukturou *struct jffs2_raw_node_ref*, která obsahuje 2 ukazatele. První ukazuje na další *raw_node* (fyzický blok) a druhý ukazuje na další blok v seznamu. Dále samozřejmě také obsahuje offset a velikost uzlu. Flash offset obsahuje 2 volné bity. Mohou být využity jako značky. Jedna určuje zastaralost uzlu a druhá zatím není využita.



Obr. 3 Logická struktura uzlů [1]

Než používat pro *next_in_ino* dynamický seznam ukončený hodnotou NULL, je lepší, když poslední položka v seznamu raw uzlu obsahuje ukazatel na strukturu *jffs2_inode_cache* pro příslušné číslo uzlu.

Tato struktura obsahuje hodnotu NULL na pozici, kde struktura *jffs2_raw_node_ref* obsahuje ukazatel na další uzel v seznamu raw uzlů. Kód, který prochází seznam pozná, že narazil na konec. Ukazatel lze pak přetypovat na vhodný typ. Ze struktury jsou načteny potřebné informace.

Pole NULL určí, že cache struktura i-uzlu je použita pouze během inicializačního skenování souborového systému.

JFFS2 používá číslo i-uzlu k vyhledání vhodné struktury *jffs2_inode_cache* v hash tabulce. Následně využije seznam uzlů k přímému přečtení každého uzlu, který spadá pod požadovaný i-uzel. Potom sestaví kompletní mapu fyzického umístění dat každého i-uzlu.

3.4 Připojování

Připojování je rozděleno na 4 části

Nejdříve je skenováno fyzické médium. Pomocí CRC v každém uzlu je kontrolována validita. Jsou alokovány reference na raw uzly. Také jsou alokovány struktury cache uzlů a následně jsou vloženy do hash tabulky. Z uzlů na flash jsou získaná data jako verze, rozsah dat v uzlu, tato data jsou uložena do paměti cache. Informace nemusí být pak znovu čteny z média.

Po skenování, kdy jsou přečteny všechny fyzické uzly a také jsou vytvořeny mapy pro každý soubor či adresář. Takže mohou být detekovány zastaralé uzly. Pro každý validní adresář či soubor je provedeno zvětšení hodnoty *nlink* v inode cache.

Třetím bodem je nalezení uzlů pro soubory či adresáře, které nemají žádné zbývající odkazy na souborový systém a smazání těchto uzlů. Pokaždé, když je smazán adresářový uzel, je procházení restartováno.

Čtvrtým bodem je odstranění dočasných informací, které již nebudou potřeba pro provoz souborového systému.

Také je první položka struktury *jffs2_inode_cache* nastavena na NULL. Tím je určeno dosažení konce v seznamu raw uzlů pro soubor či adresář, kterému odpovídá daná *jffs2_inode_cache*

3.5 Garbage collector

Po zaplnění paměti je potřeba smazat zastaralé uzly. Jakmile je načtena celá struktura uzlu, je možné nahradit zastaralý uzel jiným existujícím uzlem. Pokud je existující uzel datový, potom je načtena stránka, ve které je uzel obsahující data. Stránka je dekomprimována a následně zapsána do nového uzlu.

Mohou však nastat problémy s kompresí. Celá stránka může být výborně komprimovaná, následně však může být doprostřed strany zaspán jeden byte, který redukuje možnost komprese stránky. Když garbage collector sesbírá originální stránku, pak se může stát, že nový zapsaný uzel je větší než původní. Pak se může objevit problém s umístěním nového uzlu.

Řešením problému může být nastavení, aby 2 uzly měli stejnou verzi pole tak dlouho dokud budou mít stejná data. Když garbage collector narazí na uzel, který lze rozšířit, ale nemá dostatek místa, zkopíruje neporušený uzel a zachová originální verzi. Pak lze do uzlu zapisovat další data.

Každý blok se může nacházet v jednom ze seznamů, především v závislosti na obsahu bloku. Během běžných operací JFFS2 je většina bloků v *clean_listu* nebo *dirty_listu*, tyto seznamy představují bloky plné platných uzlů a bloků, které obsahují nejméně jeden zastaralý uzel.

Kód vykonávající garbage collector používá výše uvedené seznamy, aby rozhodl, který blok vymaže. Volba se provádí pomocí čítače *jiffies*. Když výsledek *jiffies* modulo 100 je nenulový, pak je blok vyjmut z *dirty_listu*, jinak je blok vyjmut z *clean_listu*. Prakticky to znamená, že každý stý blok je vybrán z *clean_listu*, což by mělo zajistit rovnoměrnost opotřebení.

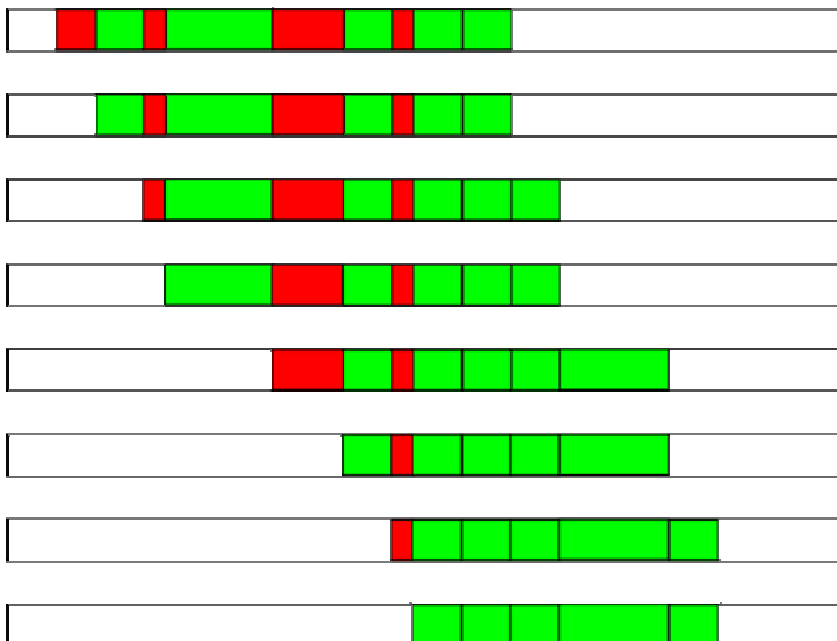
Garbage collector přesouvá platná data ze začátku záznamu na konec tak dlouho, dokud nenarazí na zastaralá data. Pak zastaralá data odstraní a pokračuje v přesouvání dalších dat.

Ukázka postupu garbage collectoru.

Před použitím garbage collectoru:



Postup garbage collectoru:



Obr. 4 Postup garbage collectoru

4 FREESCALE MQX

V této kapitole jsou popsány základní informace o operačním systému Freescale MQX a také o součástech, které jsou využity.

4.1 Základní informace

MQXTM Real-Time Operating System od MQX Embedded a ARC byl navržen pro jednoprocessorové a víceprocesorové embedded systémy reálného času.

Díky úspěšnosti operačního systému MQX firma Freescale Semiconductor využila tuto softwarovou platformu pro vlastní rodiny mikroprocesorů a to ColdFire® a PowerPCTM.

V porovnání s originálními distribucemi MQX je Freescale MQX snadněji použitelná a konfigurovatelná. MQX je run-time knihovna funkcí, které jsou využívány pro real-time a multi-tasking aplikace. Dále MQX podporuje víceprocesorové aplikace.

4.2 Ovladač FlashX

FlashX je součástí OS MQX a slouží ke komunikaci s flash zařízeními. Umožňuje získání informací o struktuře média. Médíem je myšlena interní flash paměť mikropočítače nebo externí obvody. Dále FlashX zprostředkovává zápis, čtení, přístup na libovolné místo na médiu. Také obsahuje funkce pro resetování celé flash paměti nebo jednoho sektoru.

4.2.1 FLASHX_BLOCK_INFO_STRUCT

Tato struktura obsahuje informace o fyzickém rozložení flash paměti. Jsou to následující položky:

- NUM_SECTORS [IN] — Počet sektorů o stejné velikosti v daném bloku
- START_ADDR [IN] — Počáteční adresa bloku se sektory o stejné velikosti
- SECT_SIZE [IN] — Velikost sektoru v daném bloku

Pojem sektor odpovídá pojmu paměťová buňka z 1.1

4.2.2 Další operace

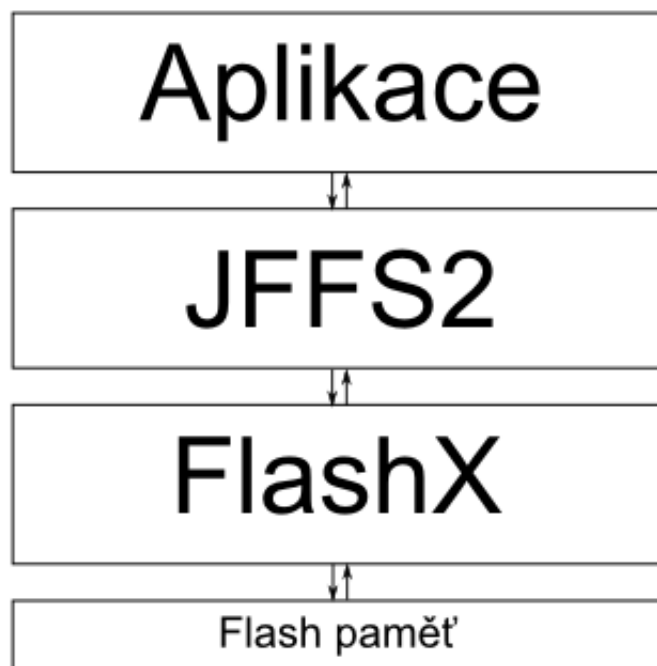
Pro čtení z média je použita funkce read() jazyka C. Podobně je pro zápis použita funkce write() a pro posun po paměti je využita funkce fseek().

II. PRAKTICKÁ ČÁST

5 NÁVRH IMPLEMENTACE

V této kapitole je popsán návrh implementace JFFS2 jako ovladače určeného pro MQX. Je vycházeno z poznatků zmíněných v teoretické části.

5.1 Schéma implementace ovladače JFFS2



Obr. 5 Schéma ovladačů

Na schématu jsou znázorněny jednotlivými bloky součásti potřebné k využití ovladače JFFS2 a také součásti, které JFFS2 využívají.

Fyzická flash paměť komunikuje s řadičem FlashX. S FlashX také komunikuje ovladač JFFS2. Aplikace využívají ovladač JFFS2 přes definované funkce. Prozatím bude využito mnou navržené rozhraní, které je níže podrobněji popsáno. Do budoucna se počítá s přepsáním do jednotného MFS (MQX File System), který používají jiné souborové systémy v MQX.

5.2 Funkce na rozhraní JFFS2 a FlashX

Funkce jsou převzaty z ukázkové aplikace pro ovladač FlashX a případně mírně upraveny. Jedná se o funkce, které přímo volají funkce ovladače FlashX a jsou využívány vyššími vrstvami ovladače JFFS2.

- `flash_open`: slouží k otevření flash paměti pro zápis nebo čtení
- `flash_close`: uzavře flash paměť
- `read_data`: načte data z požadované adresy o zadané velikosti
- `write_data`: zapíše data na požadovanou adresu paměti
- `erase_sector`: funkce resetuje blok udaný adresou

5.3 Funkce na rozhraní JFFS2 a aplikace

Při návrhu funkcí bylo vycházeno jednak z popisu JFFS2 a z práce se soubory v jazyce C.

- `jffs2_mount_flash`: připojí zvolenou flash paměť
- `jffs2_umount_flash`: odpojí zvolenou flash paměť
- `jffs2_change_dir`: funkce slouží pro vstup do zvoleného adresáře
- `jffs2_make_dir`: vytvoří nový adresář v aktuálně zvoleném adresáři
- `jffs2_remove_dir`: odstraní zvolený adresář
- `jffs2_dir`: vypíše obsah adresáře
- `jffs2_open_file`: otevře zvolený soubor pro čtení či zápis
- `jffs2_close_file`: zavře zvolený soubor, ten pak bude zapsán na disk
- `jffs2_create_file`: vytvoří soubor a zapíše do něj zvolená data
- `jffs2_write`: do otevřeného souboru zapíše zvolená data
- `jffs2_read`: z otevřeného souboru načte data dle rozsahu
- `jffs2_remove_file`: odstraní zvolený soubor
- `jffs2_init_flash`: inicializuje souborový systém na flash

5.4 Simulační prostředí

Pro rychlejší testování a menší opotřebení flash paměti na vývojovém kitu, bylo vytvořeno simulační prostředí na PC, kdy je flash nahrazena binárním souborem. Vznikly obdobné funkce pro zápis a čtení dat, jejichž funkčnost je shodná s funkcemi ovladače flashX. Také je simulováno získání informací o flash.

Kód byl vytvořen tak, aby bylo snadné přenášet kód mezi prostředím určeným k programování mikropočítače (CodeWarrior 10) a prostředím na PC (CodeLite).

K tomu slouží konstanta MQX, která není definována v případě prostředí PC. Podle toho se potom řídí překladač pomocí instrukcí `#ifdef` a `#ifndef`.

5.5 Fyzické uložení na disku

Jak bylo zmíněno v podkapitole 3.1 data jsou zapisována sekvenčně na disk.

Každý adresář a soubor jsou tvořeny dvojicí uzlů `JFFS2_NODETYPE_INODE` (dále označováno jako inode) viz. 3.2.1 a `JFFS2_NODETYPE_DIRENT` (dále označováno jako dirent) viz 3.2.2. Dient je záznam o adresáři či souboru obsahující název, číslo rodičovského adresáře, číslo adresáře či souboru. Inode obsahuje pak další informace jako např. přístupová práva, dobu vytvoření a v případě souboru i samotná data.

Jedinou výjimku tvoří kořenový adresář, který je zaznamenán pouze uzlem typu inode. Což je myslím pochopitelné, protože by nemělo být možné jej smazat nebo přejmenovat.

Uložené údaje postačují k rekonstrukci struktury adresářů a souborů.

5.6 Připojování média (Mount)

Při připojování média jsem vycházel z popisu v teoretické části 3.4 a zdrojových kódů implementace JFFS2 v eCos [5]. Během připojování je postupně skenována flash paměť a informace jsou ukládány do struktur `jffs2_raw_node_ref` a `jffs2_inocache` viz. Obr. 3 dále jsou ukládány informace o každém bloku.

Pro každý soubor či adresář je vytvořena struktura `inocache` a to podle čísla `ino` (číslo souboru, jedná-li se o soubor) nebo čísla `pino` (číslo rodičovského adresáře, jedná-li se o adresář), která obsahuje odkazy na seznam struktur `jffs2_raw_node_ref`. `Inocache` je vložena do hash tabulky (pro rychlejší vyhledávání) Tím je zajištěno, že jsou uloženy

všechny postačující informace o pozici dat na flash paměti. Takže je ušetřena paměť RAM za cenu vyššího počtu čtení z flash paměti.

Následně jsou procházeny inocache a pro každý platný adresář v rodičovské inocache je zvětšen počet nlink v inocache adresáře. Potom je jasné, které adresáře jsou validní.

V dalším průchodu jsou fyzické uzly spadající pod nevalidní adresáře označeny ke smazání.

V posledním průchodu jsou uvolněny nepotřebné informace a je zachována pouze hash tabulka struktur inocache a raw uzly.

5.7 Práce s adresáři

V této části bude základní popis operací s adresáři.

5.7.1 Výpis obsahu adresáře

Při výpisu obsahu adresáře jsou postupně procházeny záznamy spadající do inocache vypisovaného adresáře. Pro každý záznam typu dirent se získá jméno, číslo ino a číslo verze. Všechny nalezené záznamy jsou uloženy do seznamu pro kontrolu zda-li nebyly přejmenovány nebo smazány. Tento seznam je následně procházen a kontrolován. Vypíše se pouze platné položky.

5.7.2 Odstranění adresáře

Adresář daného jména je považován za smazaný, pokud je ino záznamu dirent nastaveno na 0 a číslo verze je vyšší, než číslo verze předchozího záznamu dirent. Jak již bylo zmiňováno výše data jsou vždy zapisována sekvenčně, tudíž pro smazání adresáře je potřeba vytvořit nový uzel dirent nastavit mu číslo verze o 1 vyšší a číslo ino nastavit na 0. O opravdové smazání se pak dodatečně stará garbage collector.

5.7.3 Vytvoření adresáře

Pro vytvoření nového adresáře se zadaným jménem je potřeba zapsat novou dvojici dirent inode. Číslo ino vytvářeného adresáře je získáno inkrementací o 1 dosavadního největšího čísla ino. Verze je nastavena na hodnotu 1. Číslo pino je určeno podle adresáře, ve kterém

je nový adresář vytvářen. Při vytváření adresáře je kontrolováno zda-li již adresář se stejným jménem existuje.

5.7.4 Změna aktuálního adresáře

Změna adresáře je provedena uložením čísla ino adresáře, do kterého se vstupuje a čísla ino rodičovského adresáře pro tento adresář. Při vnořování do adresářové struktury je potřeba pamatovat si informace o cestě ke kořenovému adresáři, aby pak mohlo být vstoupeno do rodiče aktuálního adresáře. Pokud je cesta pro změnu adresáře zadána absolutně je nutné, vymazat záznamy o cestě k aktuálnímu adresáři a znovu je naplnit správnými daty.

Např. Uživatel se nachází v adresáři /data/mereni a chce se přesunout do adresáře /jina_data, je nutné informace o cestě k /data/mereni vymazat, protože se pouze přidaly záznamy o cestě k /jina_data vznikla by nesmyslná cesta.

5.8 Práce se soubory

5.8.1 Vytvoření souboru

Vytváření souboru probíhá obdobně jako vytvoření adresáře, opět je potřeba zapsat dvojici dirent-inode. Taktéž je získáno nové číslo ino. Navíc mohou být zapsána data. Při vytváření souboru je kontrolováno zda-li již soubor se stejným jménem existuje.

5.8.2 Otevření souboru

Při otevření souboru zadaného jména se vytvoří popisovač souboru. Ten obsahuje inocache příslušného souboru, velikost souboru (pro kontrolu, aby uživatel nenačítal jiná data) a mód, ve kterém bude soubor otevírán (čtení, zápis). Tento popisovač bude využíván pro funkce čtení a zápisu do souboru.

5.8.3 Uzavření souboru

Uzavření souboru probíhá vynulováním dat v popisovači souboru. Soubor k uzavření je určen jeho popisovačem.

5.8.4 Zápis do souboru

Pro zápis dat do souboru je potřeba vytvořit inode s vyšší verzí než má poslední inode spadající pod daný soubor. Soubor pro zápis je určený popisovačem. Funkce pro zápis je podobná funkci *write* jazyka C.

5.8.5 Čtení ze souboru

U čtení ze souboru jsou procházeny záznamy v inocache od nejstaršího po nejnovější. Z každé inode jsou získána potřebná data spadající do zadaného rozsahu. Procházení od nejstarších záznamů po nejnovější je proto, aby případná stará data (nacházejí se ve starších záznamech) byla nahrazena novými (z novějších záznamů). Načtená data mohou být libovolného datového typu. Funkce pro čtení je podobná funkci *read* jazyka C.

5.8.6 Odstranění souboru

Soubor je stejně jako adresář odstraněn, pokud je číslo ino nastaveno na 0 a číslo verze je vyšší než u předchozího záznamu. Takže je při odstranění zapsán nový dirent s číslem ino 0 a vyšší verzí. O konečné smazání se stará garbage collector.

5.9 Odpojování média (Unmount)

Odpojení systému proběhne uvolněním struktur inocache, uvolněním popisovače souborového systému a odpojením flash paměti.

5.10 Inicializace souborového systému na médiu

Při prvním připojování je potřeba jednorázově zavolat funkci pro nastavení souborového systému flash paměti. To se provádí smazáním celé flash paměti a následným zapsáním uzlu inode s číslem ino 1 na pozici 0. Pro kořenový adresář není zapisován uzel typu dirent.

6 VLASTNÍ REALIZACE

V této části budu popisovat samotné algoritmy, které provádějí navržené postupy funkce z kapitoly 5.

6.1 Fyzické uložení na disku

Pro ukládání na fyzický disk jsem převzal z eCos [5] struktury *jffs2_raw_inode* a *jffs2_raw_dirent*. Tyto struktury patří dle mého názoru ke specifikaci souborového systému a byly navrženy autorem zdroje [1].

Struktura *jffs2_raw_inode* byla navržena takto:

```
struct jffs2_raw_inode {  
  
    uint_16 magic;    //konstanta 0x19 0x85  
  
    uint_16 nodetype; //typ JFFS2_NODETYPE_INODE  
  
    uint_32 totlen;   //celková velikost uzlu  
  
    uint_32 hdr_crc;  //crc hlavičky  
  
    uint_32 ino;      //číslo inode  
  
    uint_32 version;  //číslo verze  
  
    uint_32 mode;     //typ nebo mod souboru –zatím nevyužito  
  
    uint_16 uid;      //práva vlastníka souboru – využívá linux  
  
    uint_16 gid;      //práva skupiny – využívá linux  
  
    uint_32 isize;    //celková velikost i- uzlu  
  
    uint_32 atime;    //čas přístupu –zatím nevyužito  
  
    uint_32 mtime;    //čas modifikace –zatím nevyužito  
  
    uint_32 ctime;    //čas změny –zatím nevyužito  
  
    uint_32 offset;   //offset dat v souboru  
  
    uint_32 csize;    //komprimovaná velikost dat  
  
    uint_32 dsize;    //velikost dat po dekompresi
```

```

uint_8 compr;    //použitý algoritmus komprese – zatím nevyužito
uint_8 usercompr; //uživatelé požadovaný algoritmus – zatím nevyužito
uint_16 flags;    //označní např. zastaralého uzlu – zatím nevyužito
uint_32 data_crc; //crc komprimovaných dat – zatím nevyužito
uint_32 node_crc; //crc celého uzlu – zatím nevyužito
uint_8 data[0]; //samotná data
};

```

Struktura `jffs2_raw_dirent` byla navržena takto:

```

struct jffs2_raw_dirent{
    uint_16 magic;    //konstanta 0x19 0x85
    uint_16 nodetype; //typ JFFS2_NODETYPE_DIRENT
    uint_32 totlen;   //celková velikost uzlu
    uint_32 hdr_crc;  //crc hlavičky – zatím nevyužito
    uint_32 pino;     //číslo ino rodičovského adresáře
    uint_32 version;  //číslo verze
    uint_32 ino;      //číslo inode – pro smazání =0
    uint_32 mctime;   //-zatím nevyužito
    uint_8 nsize;     //velikost názvu disentu (bez ukončovacího znaku)
    uint_8 type;      //typ direntu (adresář: 4 soubor: 8)
    uint_8 unused[2];
    int_32 node_crc;  //crc uzlu – zatím nevyužito
    int_32 name_crc;  //crc názvu – zatím nevyužito
    uint_8 name[0];
};

```

Struktura `jffs2_raw_inode` odpovídá `JFFS2_NODETYPE_INODE` a struktura `jffs2_raw_dirent` odpovídá `JFFS2_NODETYPE_DIRENT` z podkapitoly 5.5.

6.2 Připojování (Mount)

Připojování zajišťuje funkce *jffs2_mount()*. Jediným vstupem do funkce je název flash zařízení.

Funkce nejprve vytváří strukturu *jffs2_system_info*. Tato struktura obsahuje důležité informace o souborovém systému, především tedy informace o flash paměti, dále seznam sektorů na flash (sektor je nejmenší mazatelný blok, též označovaný jako eraseblock). Dále ukazatel na volný sektor, do kterého budou zapisována data. A také seznamy různých druhů sektorů (volné, částečně zaplněné). Zatím jsem neimplementoval rozlišení na všechny druhy sektorů. Ty jsou využity především pro garbage collector.

Dále funkce *jffs2_mount()* volá funkci *mount_flash()*, která připojuje fyzickou flash paměť a také naplňuje strukturu *flashX* informacemi o flash zařízení.

Jsou tam uloženy tyto informace:

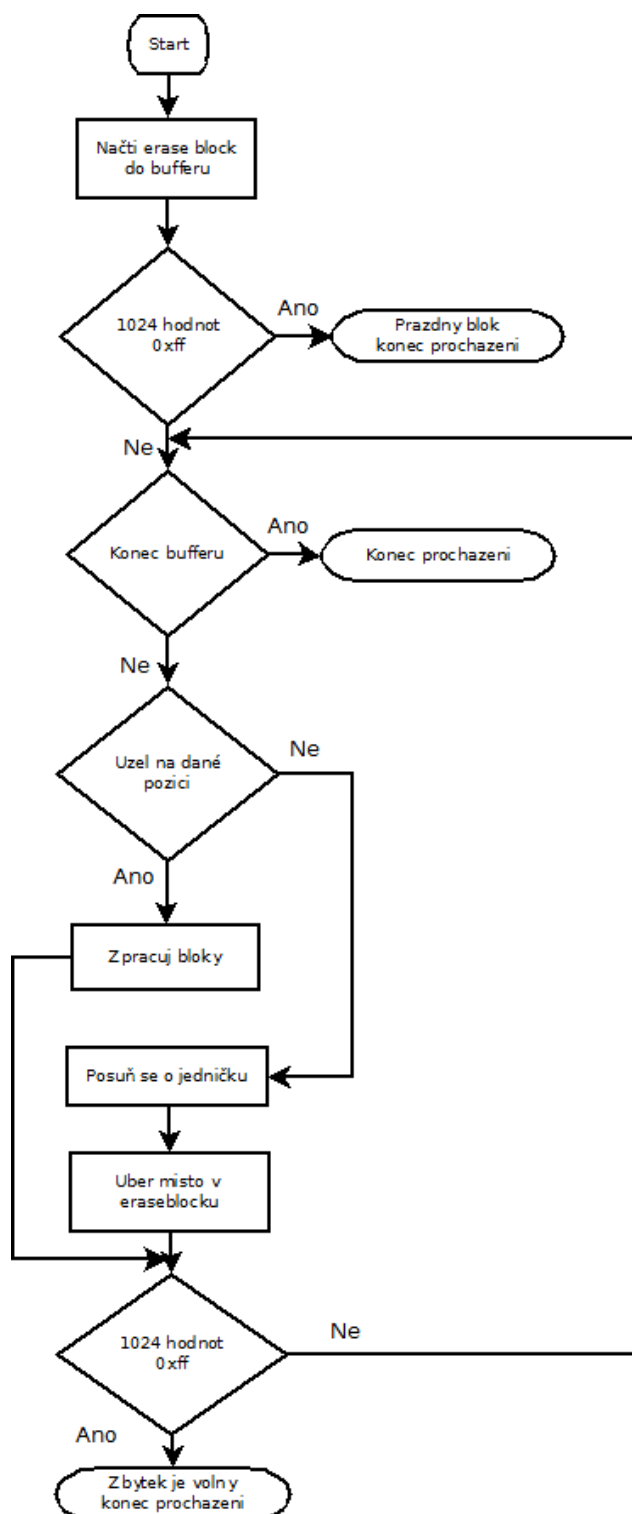
- Popisovač *flash_hdl*, využíváný k přístupu na flash
- Celkový počet bloků
- Celkový počet sektorů
- Odkaz na strukturu *info_ptr* viz. podkapitola 4.2.1

Ovladač *flashX* i simulovaný ovladač využívají funkci jazyka C *open()*.

Následně se volá funkce *jffs2_do_mount()*, která inicializuje *jffs2_system_info* a následně volá funkci *jffs2_build_file_system()*. Ta nejprve volá funkci *jffs2_scan_medium()*, která postupně prochází mazatelné bloky a vyhodnocuje jejich obsah. Zatím jsem implementoval tato vyhodnocení:

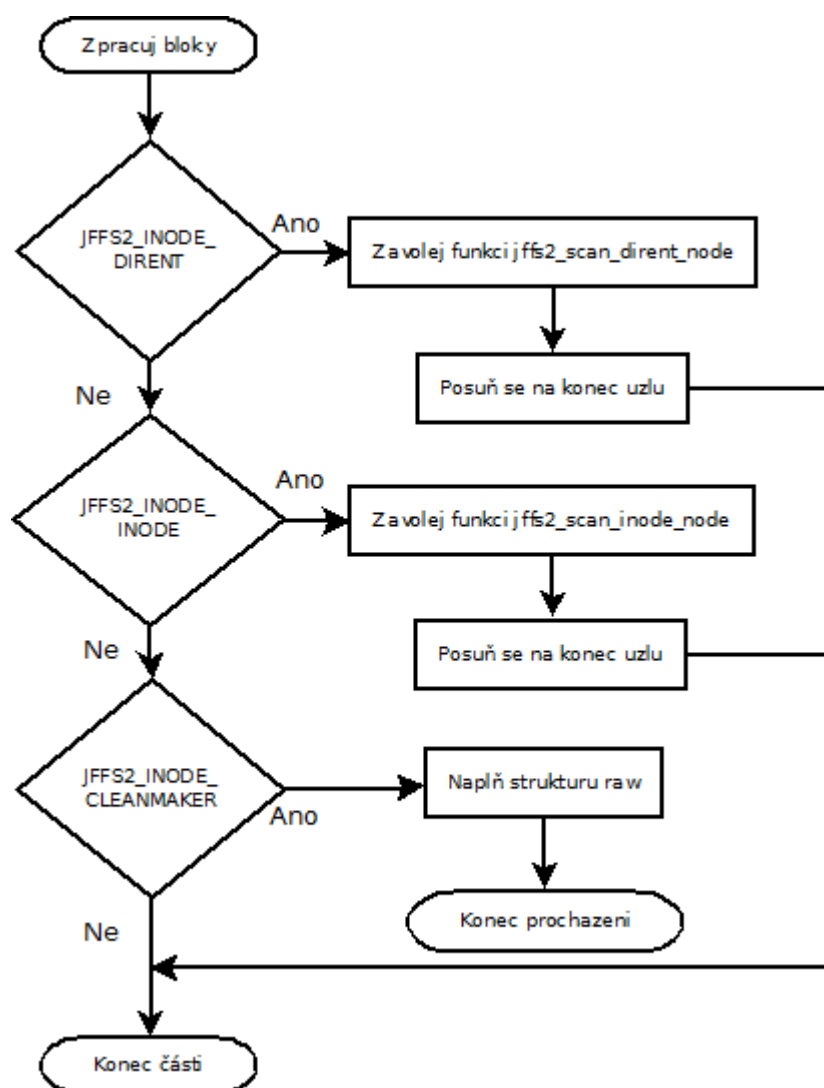
- prázdný neoznačený mazatelný blok ten je určen k resetování a označení uzlem *JFFS2_NODETYPE_CLEANMARKER* viz. podkapitola 3.2.3
- blok již označený uzlem *JFFS2_NODETYPE_CLEANMARKER*, ten je označen jako právě resetovaný, později do něj budou zapsána data
- uzel částečně zaplněný, ale ještě je volné místo pro zápis, pokud obsahuje nejvíce volného místa je zařazen na první místo pro další zápis.

Každý mazatelný blok je procházen funkcí *jffs2_scan_eraseblock()* a to následujícím postupem:



Obr. 6 Algoritmus procházení mazatelného bloku

Část zpracuj bloky se provádí následujícím způsobem (byla vyčleněna pro přehlednost diagramu).



Obr. 7 Postup části zpracuj bloky

Data potřebná k získání informací o struktuře adresářů a souborů jsou ukládána do struktur *jffs2_inode_cache* a *jffs2_raw_node_ref*. Struktury jsou vytvořeny na základě Obr. 3 a na základě eCos [5].

```
struct jffs2_inode_cache
{
    struct jffs2_full_dirent *scan_dents; //použito k dočasnému udržení seznamu
    //disentů, později přepsáno na NULL

    struct jffs2_inode_cache *next; //následující inocache v hash tabulce

    struct jffs2_raw_node_ref *nodes; // ukazatel na seznam raw uzlů

    uint_32 ino;

    _mqx_int nlink; //počet odkazů na inocache

    _mqx_int state; //zatím nevyužito
};

struct jffs2_raw_node_ref
{
    struct jffs2_raw_node_ref *next_in_ino; //Ukazuje na další raw_node_ref pro tuto
    //inocache, poslední v raw_node_ref

    //ukazuje zpět na strukturu inocache

    struct jffs2_raw_node_ref *next_phys;

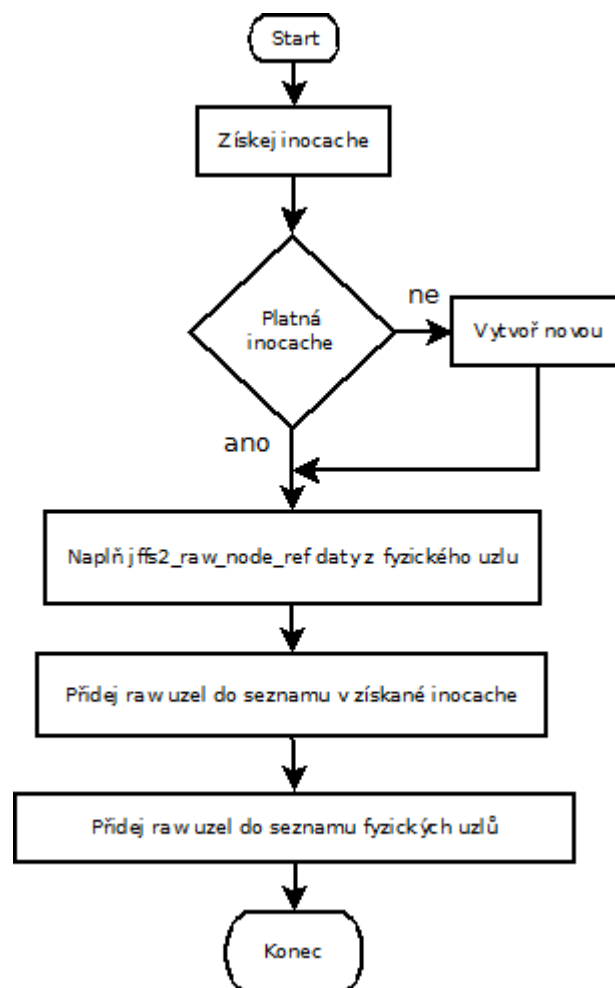
    uint_32 flash_offset; //pozice uzlu na flash paměti

    uint_32 __totlen; //velikost uzlu
};
```

Struktury inocache jsou ukládány do hash tabulky, která má pevnou velikost a je určena pro rychlejší vyhledávání podle čísla ino hledané inocache.

Ukládání do uvedených struktur provádějí funkce `jffs2_scan_dirent_node()` a `jffs2_scan_inode_node()`, které jsou popsány níže.

Funkce *jffs2_scan_inode_node()* postupuje následovně.



Obr. 8 Postup funkce *jffs2_scan_inode_node*

Činnost funkce *jffs2_scan_dirent_node()* je podobná funkci *jffs2_scan_inode_node()* s tím rozdílem, že je fyzický uzel přidáván do inocache ne podle čísla *ino*, ale podle čísla *pino* (číslo rodičovského adresáře). Dalším rozdílem je, že direnty jsou vkládány do dočasného seznamu v inocache pod položkou *scan_dents* pro pozdější použití.

Tímto je ukončeno fyzické skenování flash paměti a následují průchody jednotlivých inocache.

V prvním průchodu se v rámci každé inocache provádí funkce *jffs2_build_pass1()*.

V rámci inocache jsou zkoumány záznamy typu *dirent* a pro každý záznam je hledán odpovídající záznam typu *inode* (stejně číslo *ino*). Pokud jsou nalezeny sobě odpovídající záznamy, je zvýšena hodnota *nlink* v inocache pro záznam typu *inode*. Pokud není pro

daný záznam dirent nalezen záznam inode, pak je dirent označen jako zastaralý a určen ke smazání. Tak víme, které záznamy jsou kompletní.

V druhém průchodu jsou zpracovávány inocache, pro které nebyl nalezen v předchozím průchodu záznam dirent. Všechny záznamy *jffs2_raw_node_ref* spadající pod zpracovávanou inocache jsou označeny jako zastaralé. Pokud se jedná o neprázdný adresář, jsou rekurzivně procházeny případné další podadresáře a všechny položky jsou označovány jako zastaralé.

Ve třetím průchodu jsou uvolňovány dočasné seznamy adresářů. Na první word struktury inocache je zapsána hodnota NULL. To je proto, abychom při procházení záznamů v rámci dané inocache poznali, že jsme narazili na konec seznamu viz. Obr. 3.

6.3 Operace s uzly typu dirent

V této podkapitole budou popsány struktury a funkce, které zpracovávají a využívají uzly dirent pro operace s adresáři a soubory.

Byla definována struktura *jffs2_dirent*:

```
struct jffs2_dirent{  
  
    struct jffs2_dirent *next; //ukazatel na další položku direntu v seznamu  
  
    uint_32 pino; //odpovídá záznamu v jffs2_raw_node_dirent  
  
    uint_32 version; //odpovídá záznamu v jffs2_raw_node_dirent  
  
    uint_32 ino; //odpovídá záznamu v jffs2_raw_node_dirent  
  
    uint_32 mctime; //nevyužito  
  
    uint_8 nsize; //odpovídá záznamu v jffs2_raw_node_dirent  
  
    uint_8 type; //odpovídá záznamu v jffs2_raw_node_dirent  
  
    uchar name[0]; //odpovídá záznamu v jffs2_raw_node_dirent  
  
};
```

Do této struktury jsou načítány informace z fyzického uzlu pro pozdější zpracování.

6.3.1 Odstranění dirent

Funkce slouží k označení uzlu dirent pro smazání, to je využito pro mazání adresářů a souborů. Do funkce vstupuje instance struktura dirent a typ direntu. Je zvýšeno číslo verze a číslo ino je nastaveno na 0. Potom je zapsán nový dirent funkcí *jffs2_write_dirent*

6.3.2 Získání dirent dle cesty

Funkce slouží získání direntu z absolutně, či relativně zadané cesty. Postupně parsuje jednotlivé názvy adresářů. Pokud se jedná absolutní cestu a tato funkce je volána funkcí *jffs2_change_dir*, pak je nutné odstranit informace o cestě k adresáři, ve které se uživatel momentálně nachází (toto je podrobněji vysvětleno v kapitole 6.3.4). Pro každou část cesty je rozhodováno podle následujících pravidel jakým způsobem bude zpracována:

- Jestliže je aktuálně zpracovávaným řetězcem „..“, pak je dirent získán funkcí *jffs2_get_parent_dirent*, ta je popsána v kapitole 6.3.4
- Jinak je dirent získán funkcí *jffs2_get_dirent_by_name*, ta je popsána v kapitole 6.3.3

Když je hledaný dirent typu adresář, jsou hledány pouze direnty tohoto typu. Pokud je typ hledaného direntu soubor, nejdříve se zkusí nalézt typ adresář, jestliže není nalezen, hledá se typ soubor, protože je možné, že se jedná o poslední část cesty. Není-li nalezen během průchodu dirent požadovaného typu, je vrácena hodnota NULL.

6.3.3 Získání dirent dle názvu

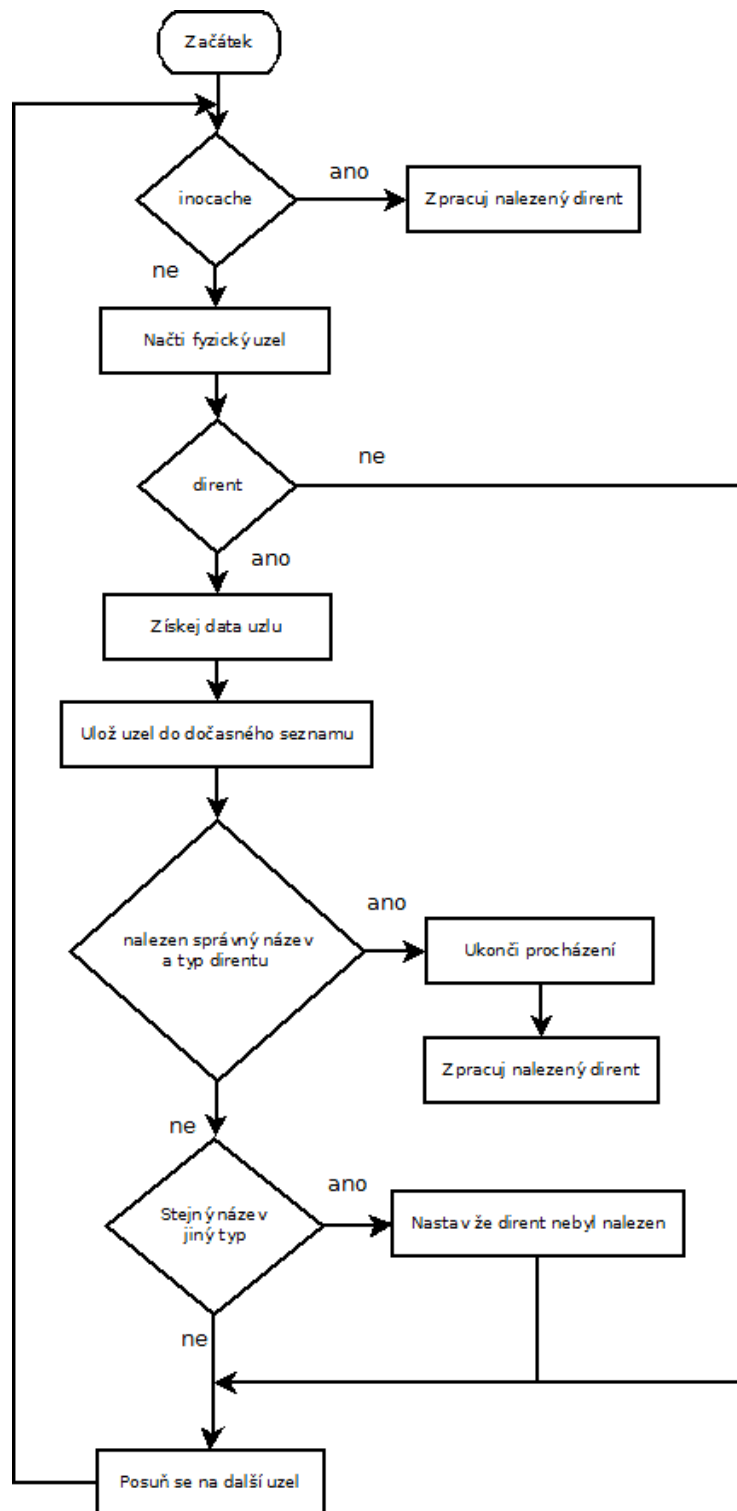
Vyhledává dirent v adresáři zadaného číslem ino. Do funkce vstupuje název direntu, který je vyhledáván, číslo adresáře, ve kterém je hledáno, typ direntu (adresář či soubor), výstupem z funkce je struktura *jffs2_dirent*.

Nejdříve je zkoumáno, zda-li se nejedná o kořenový adresář, pokud ano pak je v direntu nastaveno ino na hodnotu 1, název na hodnotu „/“.

Jinak je získána inocache prohledávaného adresáře, následně jsou procházeny všechny uzly raw spadající do této inocache a to algoritmem na Obr. 9

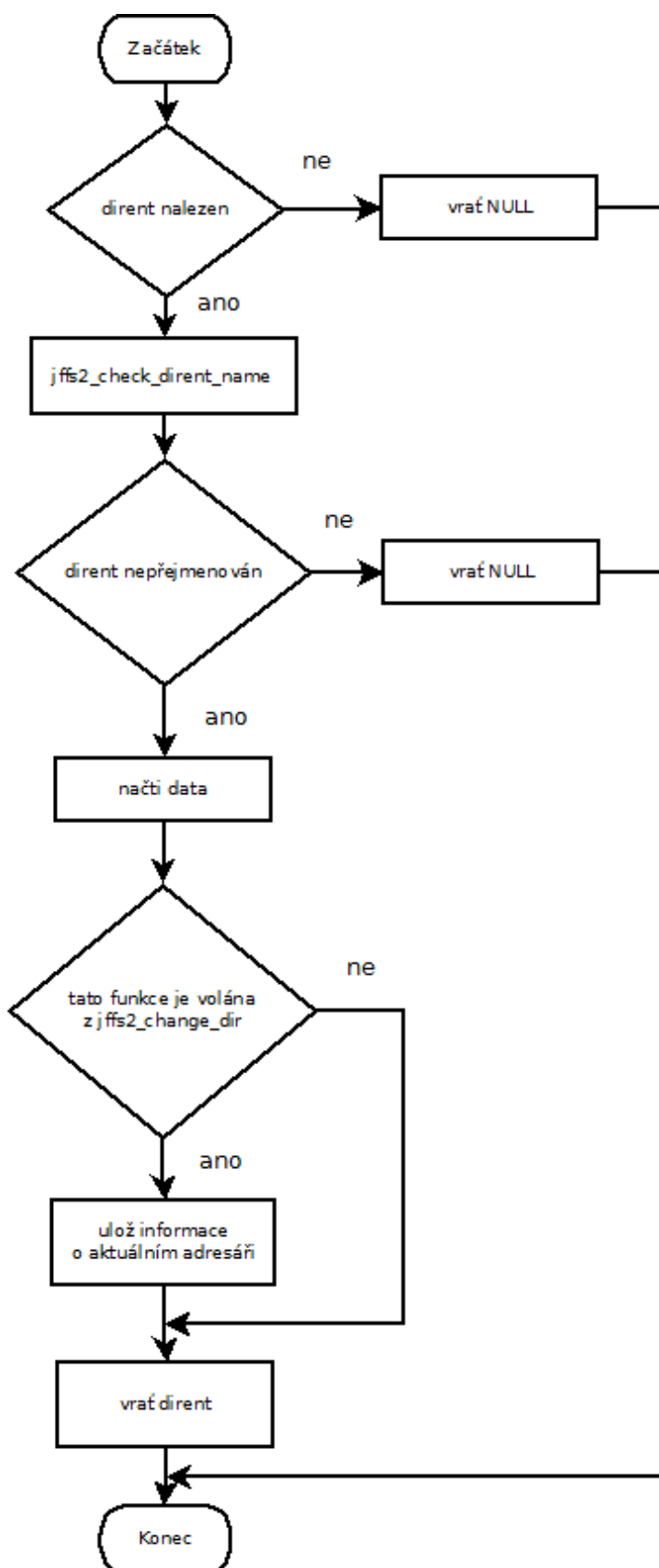
Funkce *jffs2_check_dirent_name* zkoumá zda se nevyskytuje v příslušném adresáři dirent se stejným číslem *ino*, vyšší verzí než je nejvyšší verze nalezeného direntu a jiným jménem.

Pokud ano vrací se hodnota 1, kterou je určeno, že byl dirent přejmenován.



Obr. 9 Algoritmus funkce *jffs2_get_dirent_by_name*

Část: Zpracuj nalezený dirent popisuje následující diagram



Obr. 10 Část zpracuj nalezený dirent

6.3.4 Získání dirent rodičovského adresáře

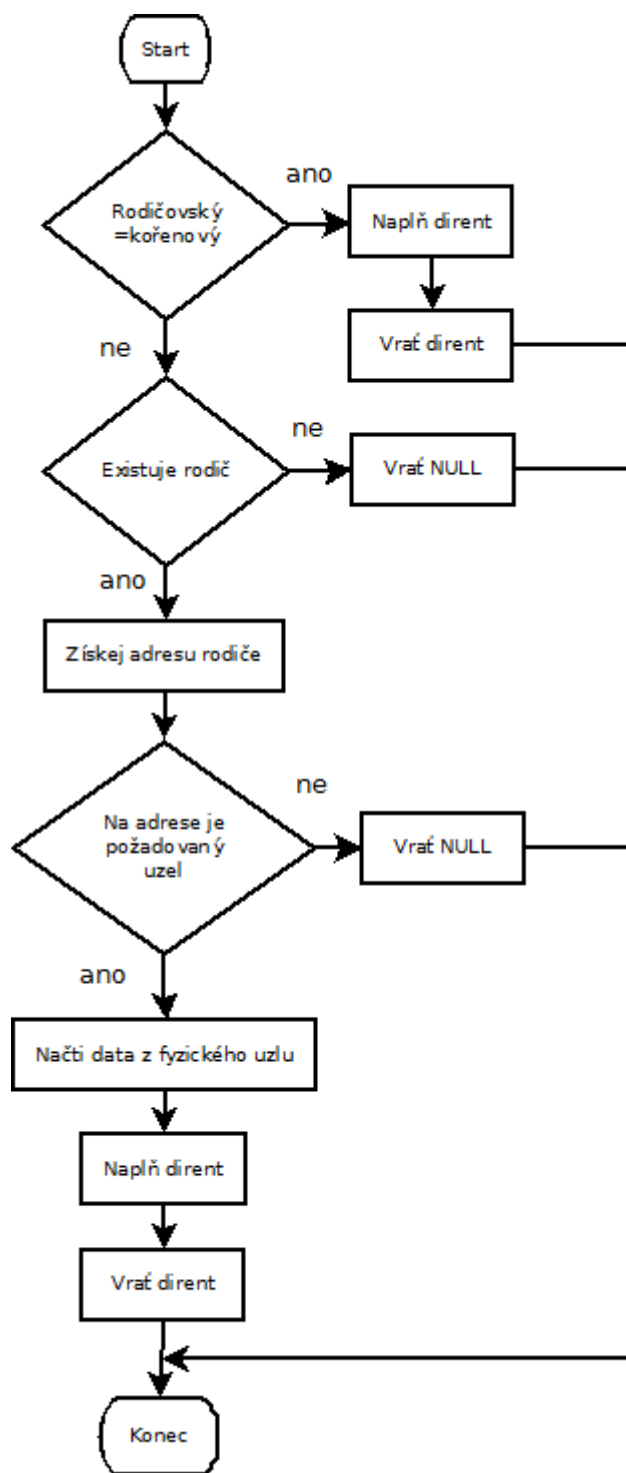
Funkce slouží k vyhledání rodičovského adresáře aktuálního adresáře, je využívána především v případě když je součástí cesty řetězec „..“, což značí posun o adresář výše.

Ve funkcích *jffs2_get_dirent_by_name* a *jffs2_get_dirent_by_path* je při každém vstupu hlouběji do stromu adresářů zaznamenán do struktury *jffs2_dirent_offset* offset opouštěného adresáře a ukazatel na předchozí strukturu stejného typu, to pak umožňuje nacházet cestu zpět ke kořeni. Pokud je při změně adresáře cesta zadána absolutně nebo je aktuálním adresářem kořenový adresář, je potřeba vymazat seznam záznamů o cestě z aktuálního adresáře ke kořenu. Seznam je potom s vnořováním dle dané cesty opět získán.

Struktura *jffs2_dirent_offset* je definována takto:

```
struct jffs2_dirent_offset
{
    struct jffs2_dirent_offset *prev; //ukazatel adresu rodičovského adresáře
    uint_32 offset; //pozice aktuálního adresáře na flash
};
```

Algoritmus získání rodičovského adresáře je popsán na Obr. 11



Obr. 11 Algoritmus funkce získání
rodičovského adresáře

6.4 Operace s uzly typu inode

V této podkapitole jsou popsány funkce, které pracují s uzly typu `JFFS2_NODETYPE_INODE`, tedy uzly obsahující data souborů. Funkce jsou využity především pro načítání informací o souborech a vlastních dat.

Inode jsou ukládány do struktury `jffs2_inode`. Tato struktura obsahuje stejné informace jako struktura `jffs2_raw_inode`, kromě vlastních dat souboru a úvodní hlavičky.

6.4.1 Získání seznamu uzlů inode

Funkce slouží k získání všech uzlů inode spadajících pod soubor, uzly jsou uloženy do seznamu a to od nejstarší verze po nejnovější. To se děje kvůli usnadnění rekonstrukce souboru při čtení z něj. Do funkce vstupuje `inocache` souboru, výstupem je dynamický seznam inode.

6.4.2 Získání inode s nejvyšším číslem ino

Funkce slouží k získání inode s nejvyšším číslem verze, do funkce vstupuje seznam inode pro daný soubor, výstupem z funkce je struktura `jffs2_inode`. Funkce postupně prochází seznam inode a pokud má aktuální inode vyšší číslo, než je nejvyšší dosažené, je uložena jako inode s nejvyšším číslem.

6.4.3 Získání velikosti souboru

Funkce zjišťuje velikost souboru, ta je určena pozicí dat v souboru + velikost dat. Postupně je procházen seznam inode a pokud je velikost souboru pro danou inode větší, než aktuální velikost souboru, pak je nová velikost uložena jako největší. Do funkce vstupuje seznam inode, výstupem je velikost souboru.

6.4.4 Načtení vlastních dat z inode

Funkce načítá potřebná data z inode, aby pak mohla být zrekonstruována požadovaná data souboru. Do funkce vstupuje inode, ze které se data načítají, dále pozice dat v souboru, velikost načítaných dat a buffer, do kterého jsou získaná data zapisována.

Nejprve je zjištěno, zda-li vůbec inode obsahuje, alespoň část hledaných dat, pokud ne, je vrácen původní buffer.

Pokud ano jsou data získána v případě jedné ze situací

- Offset v inode je větší než požadovaný offset a zároveň velikost požadovaných dat přesahuje velikost dat v inode. V tom případě jsou načtena pouze data v inode. Například chceme načíst data na offsetu 0 o velikosti 5, v inode je offset 2 a velikost dat je 2, potom jsou načtena pouze data na pozici 2-4
- Offset v inode je větší než požadovaný offset. V tom případě jsou načtena data od offsetu v inode až po zadanou velikost. Například chceme načíst data na offsetu 0 o velikosti 5, v inode je offset 2 a velikost dat je 4, potom jsou načtena data na pozici 2-5.
- Velikost požadovaných dat přesahuje velikost dat v inode. V tom případě jsou načtena data od požadovaného offsetu po konec dat v inode. Například chceme načíst data na offsetu 0 o velikosti 5, v inode je offset 0 a velikost dat je 3, potom jsou načtena data na pozici 0-3.
- Ani jeden z předcházejících případů. Potom jsou načítána data ze zadaného offsetu o zadané velikosti. Například chceme načíst data na offsetu 0 o velikosti 5, v inode je offset 0 a velikost dat je 7, potom jsou načtena data na pozici 0-5.

Po získání dat z inode, jsou data zapsána do výstupního bufferu.

6.5 Funkce pro zápis fyzických uzlů

V této podkapitole budou popsány funkce, které zajišťují zápis na fyzické médium. Tyto funkce jsou využívány pro vytváření, mazání adresářů a souborů a také pro zápis do souboru.

6.5.1 Zápis fyzického uzlu *raw_dirent*

Funkce zapisuje strukturu *jffs2_raw_dirent* na flash. Do funkce vstupuje struktura *jffs2_dirent*, ze které jsou získány potřebné informace pro vytvoření struktury *jffs2_raw_dirent*, která je pak zapsána funkcí z ovladače flashX.

6.5.2 Zápis fyzického uzlu *raw_inode*

Funkce zapisuje strukturu *jffs2_raw_inode* na flash. Do funkce vstupuje struktura *jffs2_inode*, ze které jsou získány potřebné informace pro vytvoření struktury *jffs2_raw_inode*. Dalším vstupem je ukazatel na zapisovaná data. Pokud je ukazatel na data roven hodnotě NULL (to je v případě vytváření adresáře), pak nejsou na flash zapisovány informace o velikosti dat a celková velikost uzlu je určena velikostí struktury *jffs2_raw_inode*. Jestliže jsou ukládána i data (vytváření, či zápis do souboru), pak jsou uvedeny informace o velikosti dat a celková délka uzlu je určena velikostí struktury *jffs2_raw_inode* + velikost dat. Vytvořená struktura je pak zapsána funkcí z ovladače flashX.

6.5.3 Zápis logického uzlu *dirent*

Funkce zajišťuje vyhledání místa pro zápis direntu a aktualizaci informací o struktuře uzlů na flash. Nejdříve je funkcí *jffs2_get_offset_to_write* (viz. 0) získána pozice pro zápis fyzického uzlu, následně je funkcí *jffs2_write_raw_dirent* zapsán uzel na flash.

Potom je funkcí *jffs2_take_off_free_space* aktualizováno volné místo v zapisovaném eraseblocku. Dále je právě zapsaný dirent vložen do struktury *jffs2_raw_node_ref* a ta je vložena do seznamu v *jffs2_inode_cache*, to se děje proto, aby bylo možné hned se zapsaným uzlem pracovat. Pak je uzel vložen do seznamu fyzických uzlů v daném eraseblocku. Pokud vše proběhlo v pořádku, funkce vrací hodnotu 0.

6.5.4 Zápis logického uzlu *inode*

Funkce provádí podobný postup jako funkce *jffs2_write_dirent* s tím rozdílem, že slouží k zápisu *inode*. Dalším rozdílem je výpočet zbývajících volného místa v eraseblocku. Pokud zapisovaná *inode*, která neobsahuje data je ubráno místo pouze o velikosti struktury *jffs2_raw_inode*, jinak je místo spočítáno jako velikost struktury *jffs2_raw_inode* + velikost dat.

6.5.5 Získání pozice pro zápis dat

Funkce zjišťuje pozici na, kterou je možné zapisovat fyzický uzel. Nejprve je získán blok určený pro zápis. Následně je podle pozice bloku na flash paměti získána jeho velikost.

Pokud již není místo v aktuálním bloku, je zvolen další blok s dostatkem místa a je uložen jako blok vhodný pro zápis. Nakonec je spočítána a vrácena pozice vhodná pro zápis.

6.6 Operace s adresáři

V této podkapitole budou popsány funkce, které zpracovávají adresáře. Tyto funkce bude volat uživatel. Do všech funkcí vstupuje název adresáře. Cesta k němu může být uvedena relativně nebo absolutně.

6.6.1 Změna adresáře

Funkce slouží ke změně aktuálního adresáře. Nejdříve je nastavena proměnná *change* ve struktuře *jffs2_system_info* na jedničku, aby byla zachována cesta ke kořenovému adresáři.

Dirent je získán funkcí *jffs2_get_dirent_by_path* popsanou v kapitole 6.3.2. Pokud dirent pro hledaný adresář neexistuje, je vrácena chyba. Jinak jsou nastaveny informace o aktuálním adresáři ve struktuře *jffs2_system_info*. Potom je funkce ukončena.

6.6.2 Odstranění adresáře

Funkce slouží ke smazání adresáře. Nejdříve získá dirent pro zadané jméno. Potom zavolá funkci *jffs2_remove_dirent* popsanou v kapitole 6.3.1. Pokud proběhne vše v pořádku, funkce vrací hodnotu 0.

6.6.3 Vytvoření adresáře

Funkce vytváří nový adresář. Nejdříve je zadaný řetězec rozdělen na cestu a na název vytvářeného adresáře. Potom je zkontrolováno, zda-li zadaný adresář již existuje. Pokud neexistuje, je získána cesta k adresáři, ve kterém má být nový adresář vytvořen. Následně je vytvořena struktura dirent s novým číslem *ino* a je zapsána na flash paměť funkcí *jffs2_write_dirent*, popsanou v kapitole 6.5.3. Obdobně je vytvořena struktura inode, která je také zapsána na flash paměť a to funkcí *jffs2_write_inode*, popsanou v kapitole 6.5.4.

Pokud vše proběhlo v pořádku, funkce vrací hodnotu 0.

6.6.4 Výpis obsahu adresáře

Funkce slouží k výpisu obsahu adresáře. Nejprve je zjištěno zda-li není vstupem řetězec „.“, pokud ano, je vypsán obsah aktuálního adresáře, jinak je získán dirent funkcí *jffs2_get_dirent_by_path*, popsanou v kapitole 6.3.3. Pak je vypsán obsah adresáře dle získaného direntu. Výpis obsahu adresáře provádí funkce *jffs2_dir_content*.

V této funkci je nejdříve získána inocache, následně jsou procházeny uzly dirent, které pod ni spadají a jsou ukládány do seznamu, také je zjištěno nejvyšší číslo ino pro položky v adresáři. Pokud adresář neobsahuje žádné položky, je funkce ukončena. Dále je prováděn algoritmus, který postupně pro každé ino záznamů nacházejících se v inocache dělá následující kroky. V prvním průchodu seznamu direntů je pro ino i hledán nejnovější záznam. V druhém průchodu je kontrolováno zda-li se v seznamu nenachází záznam se stejným jménem ale číslem ino 0 (to značí smazaný dirent). Priority slouží k rozlišení novějšího záznamu. Pokud je dirent označen jako smazaný a jedná se nejnovější záznam, pak není vypsán. Podrobnější algoritmus je zobrazen v příloze P I.

6.7 Operace se soubory

V této podkapitole budou popsány funkce a struktury, které provádějí operace se soubory. Tyto funkce bude volat uživatel. Do všech funkcí vstupuje název souboru. Cesta k němu může být uvedena relativně nebo absolutně.

Pro práci se soubory je vytvářen popisovač souboru. Jedná se o strukturu *jffs2_file_id*.

```
struct jffs2_file_id
{
    struct jffs2_inode_cache *jic; // inocache pro dany soubor
    uint_32 open_mode; //mod otevreni souboru (cteni, zapis)
    uint_32 file_size; //velikost souboru
};
```

6.7.1 Otevření souboru

Funkce vytváří popisovač souboru, se kterým je nadále pracováno v ostatních funkcích. Vstupními parametry jsou název otevíraného souboru a mód pro otevření. Nejdříve je

zjištěna existence souboru, pokud soubor neexistuje je funkce ukončena. Jinak je získána inocache zadaného souboru. Pak je naplněna struktura *jffs2_file_id*. Velikost souboru je získána funkcí *jffs2_get_file_size* (viz.6.3.4). Pokud vše proběhlo v pořádku, je vrácen popisovač souboru.

6.7.2 Zavření souboru

Do funkce vstupuje popisovač uzavíraného souboru. Data tohoto popisovače jsou vynulována. Tím pádem pomocí daného popisovače nebude možné provádět operace se souborem, ke kterému popisovač náležel.

6.7.3 Zápis do souboru

Funkce zapisuje data do souboru. Soubor je určen jeho popisovačem. Dalšími vstupy jsou zapisovaná data, pozice, na kterou mají být data zapsána a velikost zapisovaných dat. Nejprve je ověřeno zda-li je soubor otevřen ve správném módu. Dále je ověřeno zda-li zadaná pozice nepřesahuje velikost souboru (tím pádem by vznikla v souboru mezera bez dat). Potom je získána z popisovače souboru inocache. Následně je vytvořen seznam inode (funkce *jffs2_get_inode_list* viz.6.4.1). Ze seznamu inode je získána inode s nejvyšším číslem verze (funkce *jffs2_get_inode_high_ver* viz.6.4.2). V této inode je pak číslo verze zvětšeno o jedničku a inode je společně s daty zapsána funkcí *jffs2_write_inode* (funkce je popsána v podkapitole 6.5.4). Pokud zápis proběhl v pořádku je vrácena 0.

6.7.4 Čtení ze souboru

Funkce čte data ze souboru. Soubor je určen jeho popisovačem. Dalšími vstupy jsou pozice, ze které mají být data načítána, velikost čtených dat a buffer, do kterého budou načtená data zapsána. Nejprve je ověřeno zda-li je soubor otevřen ve správném módu. Dále je ověřeno zda-li zadaná pozice a velikost nepřesahuje velikost souboru (pak by byla načtena nesprávná data). Potom je získána z popisovače souboru inocache. Následně je vytvořen seznam inode (funkce *jffs2_get_inode_list* viz.6.4.1). Potom je postupně procházen seznam inode a funkcí *jffs2_read_inode_data* jsou rekonstruována data ze souboru. Po projití celého seznamu je naplněn výstupní buffer. Pokud vše proběhlo v pořádku funkce vrací 0.

6.7.5 Vytvoření souboru

Funkce vytváří nový soubor a případně zapisuje zadaná data. Nejdříve je zadaný řetězec rozdělen na cestu a název vytvářeného souboru. Potom je zkontrolováno zda-li zadaný adresář již existuje. Pokud neexistuje, je získána cesta k adresáři, ve kterém má být nový soubor vytvořen. Následně je vytvořena struktura dirent s novým číslem ino a je zapsána na flash paměť funkcí *jffs2_write_dirent*, popsanou v kapitole 6.5.3. Obdobně je vytvořena struktura inode, která je taktéž zapsána na flash paměť a to funkcí *jffs2_write_inode*, popsanou v kapitole 6.5.4.

Pokud vše proběhlo v pořádku funkce vrací hodnotu 0.

6.7.6 Odstranění souboru

Funkce odstraní soubor. Podobně jako u odstranění adresáře je nejprve zkontrolována existence adresáře a následně je zavolána funkce *jffs2_remove_dirent*. Pokud vše proběhlo v pořádku je vrácena hodnota 0.

6.8 Odpojování (Unmount)

Připojování zajišťuje funkce *jffs2_unmount()*. Jediným vstupem do funkce je popisovač souborového systému. Během odpojování jsou postupně uvolňovány uzly ve strukturách inocache. Dále je uvolněn popisovač souborového systému a je odpojeno zařízení flash.

6.9 Inicializace zařízení

Při inicializaci je nejdříve získán popisovač flash zařízení. Následně je vymazána celá flash paměť (to provádí interní funkce ovladače flashX). Po smazání flash paměti je zapsána struktura *jffs2_raw_inode*, která neobsahuje žádná další data, číslo ino je nastaveno na 1 a číslo verze na 1. Nakonec je flash odpojena. Toto provádí funkce *jffs2_init_flash*.

7 PŘEHLED SOUBORŮ OVLADAČE

Zde je uveden seznam všech souborů ovladače JFFS2 vytvořených v rámci diplomové práce.

- jffs2.h – prototypy funkcí pro komunikaci s aplikacemi
- jffs2.c – vlastní těla funkcí pro komunikaci s aplikacemi
- build.c – funkce pro připojení flash paměti, vyhodnocení obsahu a uložení informací do struktury *jffs2_system_info*
- directory.c – pomocné funkce pro práce s adresáři
- dirent.c – funkce pro operaci s uzly dirent
- flashx.h (MQX_flashx.h) – prototypy funkcí pro operace s flash pamětí, definice struktury s informacemi o flash paměti. Byly vytvořeny dvě verze souborů - simulační soubor pro ladění na PC, flashx.h a soubor pro použití v reálné aplikaci, MQX_flashx.h.
- flashx.c (MQX_flashx.c) – vlastní těla funkcí pro operace s flash pamětí. Stejně jako u předchozího souboru existují dvě verze.
- ino_cache.c – funkce pro práci se strukturami *jffs2_inode_cache*, načítání nebo vkládání do hash tabulky
- inode.c – funkce pro operace s uzly inode
- list.h – definice seznamů, prototypy funkcí
- list.c – funkce pro operace se seznamy, zatím pouze přidávání do seznamu
- nodes.h – definice struktur logických uzlů a prototypy vnitřních funkcí ovladače
- psptypes.h – definice datových typů MQX, použito pouze při simulaci na PC
- raw_nodes.h – definice struktur fyzických uzlů
- scan.c – funkce pro načtení obsahu mazatelného bloku
- system_info.h – definice struktury pro ukládání informací o systému
- write.c – funkce pro zápis uzlů na flash

8 TESTOVÁNÍ

V této kapitole bude popsáno testování navržených funkcí. Jejich správnost byla ověřována jednak v simulačním prostředí na PC, kde šel snadno zjistit obsah simulované flash, tak i na vývojovém kitu, kde jsou k dispozici pouze kontrolní výpisy. Postupně budou uváděny výsledky prováděných funkcí.

8.1 Inicializace souborového systému pro médium

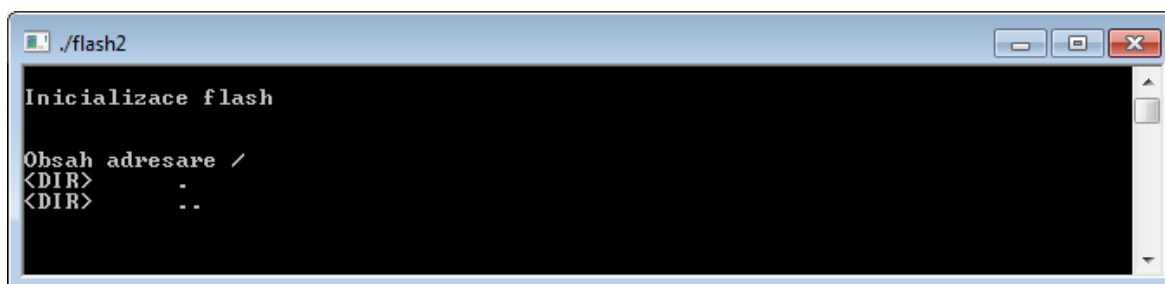
Inicializace je provedena resetováním celé flash paměti a zápisem údaje o kořenovém adresáři. Na obrázku Obr. 12 je vidět uzel indikující kořenový adresář.

```
00000000: 85 19 02 E0 44 00 00 00|00 00 00 00 01 00 00 00 | ...D.....
00000010: 01 00 00 00 00 00 00 00|F4 01 F4 01 00 00 00 00 | .....ôô.....
00000020: 00 00 00 00 00 00 00 00|00 00 00 00 00 00 00 00 | .....
00000030: 00 00 00 00 00 00 00 00|00 00 00 00 00 00 00 00 | .....
00000040: 00 00 00 00 FF FF FF FF|FF FF FF FF FF FF FF FF | .....
00000050: FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF | .....
00000060: FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF | .....
00000070: FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF | .....
00000080: FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF | .....
00000090: FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF | .....
000000A0: FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF | .....
000000B0: FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF | .....
000000C0: FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF | .....
000000D0: FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF | .....
000000E0: FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF | .....
```

Obr. 12 Uzel kořenového adresáře

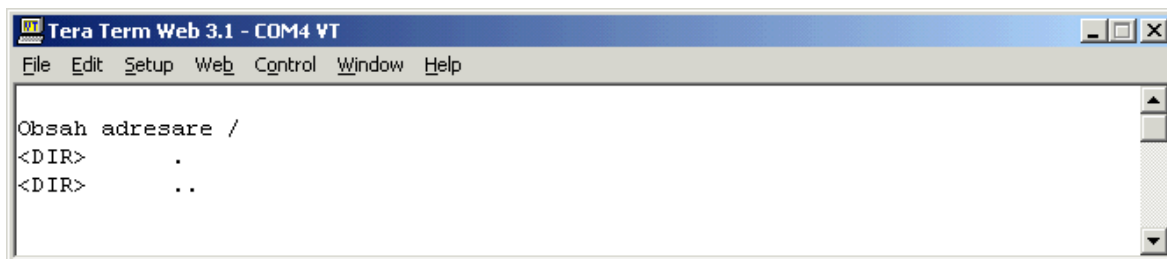
Po inicializaci flash paměti a připojení byl vypsán pro kontrolu obsah kořenového adresáře.

Jak je patrné z obrázku Obr. 13, kořenový adresář neobsahuje žádné soubory a adresáře.



Obr. 13 Kontrolní výpis kořenového adresáře

Na obrázku Obr. 14 je kontrolní výpis z terminálu při testování na vývojovém kitu.



Obr. 14 Výpis pro testování na kitu

8.2 Vytvoření adresáře nebo souboru

Adresář nebo soubor je zapsán pomocí dvojice dirent - inode. Testovací funkce vytvořila adresář se jménem adresar a to v kořenovém adresáři. Na obrázku Obr. 15 jsou označeny zapsané uzly.

```
00000000: 85 19 02 E0 44 00 00 00|00 00 00 00 01 00 00 00 | ...00rD00000000
00000010: 01 00 00 00 00 00 00 00|F4 01 F4 01 00 00 00 00 | 00000000ôô000000
00000020: 00 00 00 00 00 00 00 00|00 00 00 00 00 00 00 00 | 0000000000000000
00000030: 00 00 00 00 00 00 00 00|00 00 00 00 00 00 00 00 | 0000000000000000
00000040: 00 00 00 00 85 19 01 E0|30 00 00 00 00 00 00 00 | 0000...00r0000000
00000050: 01 00 00 00 01 00 00 00|02 00 00 00 00 00 00 00 | 0000000000000000
00000060: 07 04 00 00 00 00 00 00|00 00 00 00 00 61 64 72 65 | 000000000000adre
00000070: 73 61 72 00 85 19 02 E0|44 00 00 00 00 00 00 00 | sarD...00rD000000
00000080: 02 00 00 00 01 00 00 00|00 00 00 00 F4 01 F4 01 | 000000000000ôô
00000090: 00 00 00 00 00 00 00 00|00 00 00 00 00 00 00 00 | 0000000000000000
000000A0: 00 00 00 00 00 00 00 00|00 00 00 00 00 00 00 00 | 0000000000000000
000000B0: 00 00 00 00 00 00 00 00|FF FF FF FF FF FF FF FF | 00000000'.....
000000C0: FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF | .....
000000D0: FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF | .....
000000E0: FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF | .....
000000F0: FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF | .....
```

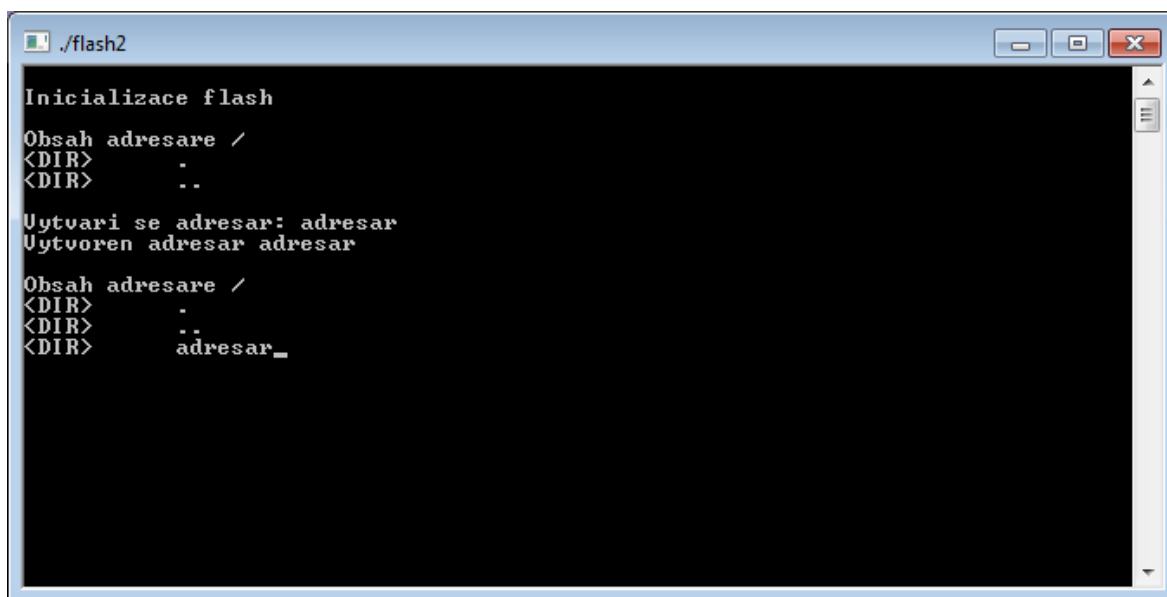
Začátek uzlu kořenového adresáře

Začátek uzlu dirent adresáře adresar

Začátek uzlu inode adresáře adresar

Obr. 15 Uzly na flash médiu po zapsání adresáře

Na obrázku Obr. 16 je pak ve výpisu kořenového adresáře zobrazen zapsaný adresář.



```
./flash2

Inicializace flash

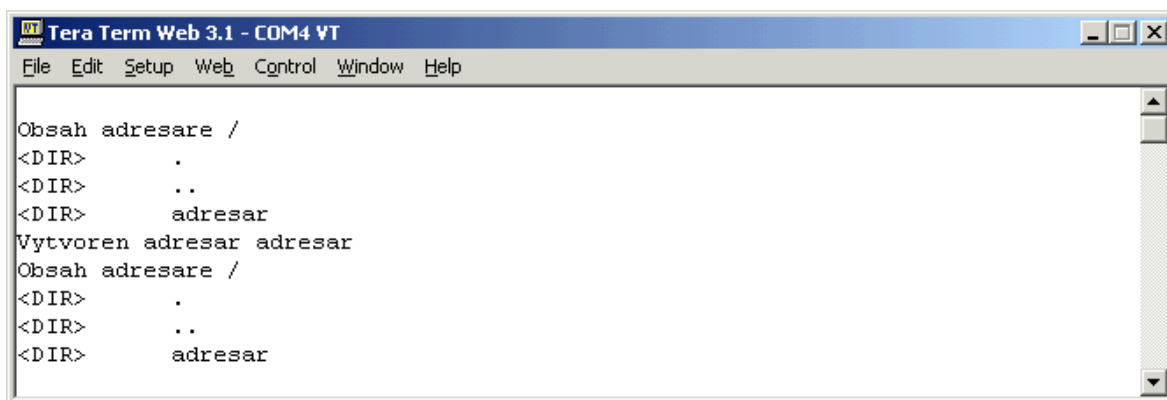
Obsah adresare /
<DIR>      .
<DIR>      ..

Uytvari se adresar: adresar
Uytvoren adresar adresar

Obsah adresare /
<DIR>      .
<DIR>      ..
<DIR>      adresar_
```

Obr. 16 Zapsání adresáře a kontrolní výpis

Obrázek Obr. 17 obsahuje kontrolní výpis při testování na vývojové desce



```
Tera Term Web 3.1 - COM4 VT
File Edit Setup Web Control Window Help

Obsah adresare /
<DIR>      .
<DIR>      ..
<DIR>      adresar
Vytvoren adresar adresar
Obsah adresare /
<DIR>      .
<DIR>      ..
<DIR>      adresar
```

Obr. 17 Výpis u vytvoření adresáře

8.3 Zápis do souboru

Při zápisu do souboru je zapsán nový záznam inode s aktuálními daty. Na obrázku Obr. 18 jsou ukázány původní uzly souboru a také nový uzel.

```

00000000: 85 19 02 E0 44 00 00 00|00 00 00 00 01 00 00 00 | ...0rD000000000000
00000010: 01 00 00 00 00 00 00 00|F4 01 F4 01 00 00 00 00 | 000000000000000000
00000020: 00 00 00 00 00 00 00 00|00 00 00 00 00 00 00 00 | 000000000000000000
00000030: 00 00 00 00 00 00 00 00|00 00 00 00 00 00 00 00 | 000000000000000000
00000040: 00 00 00 00 85 19 01 E0|30 00 00 00 00 00 00 00 | 0000...0r0000000000
00000050: 01 00 00 00 01 00 00 00|02 00 00 00 00 00 00 00 | 000000000000000000
00000060: 07 04 00 00 00 00 00 00|00 00 00 00 61 64 72 65 | 000000000000adre
00000070: 73 61 72 00 85 19 02 E0|44 00 00 00 00 00 00 00 | sar0...0rD000000000
00000080: 02 00 00 00 01 00 00 00|00 00 00 00 F4 01 F4 01 | 000000000000000000
00000090: 00 00 00 00 00 00 00 00|00 00 00 00 00 00 00 00 | 000000000000000000
000000A0: 00 00 00 00 00 00 00 00|00 00 00 00 00 00 00 00 | 000000000000000000
000000B0: 00 00 00 00 00 00 00 00|85 19 01 E0 33 00 00 00 | 00000000...0r3000
000000C0: 00 00 00 00 01 00 00 00|01 00 00 00 03 00 00 00 | 000000000000000000
000000D0: 00 00 00 00 00 0A 08 00|00 00 00 00 00 00 00 00 | 000000000000000000
000000E0: 00 73 6F 75 62 6F 72 2E|74 78 74 85 19 02 E0 4E | 0soubor.txt...0rN
000000F0: 00 00 00 00 00 00 00 00|00 00 00 01 00 00 00 00 | 000000000000000000
00000100: 00 00 00 F4 01 F4 01 0D|0A 00 00 00 00 00 00 00 | 000000000000000000
00000110: 00 00 00 00 00 00 00 00|00 00 00 00 00 0A 00 00 | 000000000000000000
00000120: 00 0D 0A 00 00 00 00 00|00 00 00 00 00 00 00 61 | 0000000000000000a
00000130: 68 6F 6A 20 73 76 65 74|65 85 19 02 E0 4F 00 00 | hoj svete...0r000
00000140: 00 00 00 00 00 03 00 00|00 02 00 00 00 00 00 00 | 000000000000000000
00000150: 00 F4 01 F4 01 08 00 00|00 00 00 00 00 00 00 00 | 000000000000000000
00000160: 00 00 00 00 00 00 00 00|00 0B 00 00 00 0B 00 00 | 000000000000000000
00000170: 00 00 00 00 00 00 00 00|00 00 00 00 00 68 65 6C | 00000000000000hel
00000180: 6C 6F 20 77 6F 72 6C 64|FF FF FF FF FF FF FF FF | lo world'.....
00000190: FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF | .....
000001A0: FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF | .....
000001B0: FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF | .....

```

Začátek uzlu dirent pro soubor.txt

Začátek uzlu inode pro soubor.txt verze 1

Začátek uzlu inode pro soubor.txt verze 2

Obr. 18 Uzly po zápisu do souboru

Na obrázku Obr. 19 je pak pro kontrolu vypsán obsah souboru před zápisem a po zápisu.

```

./flash2

Obsah souboru soubor.txt
ahoj svete

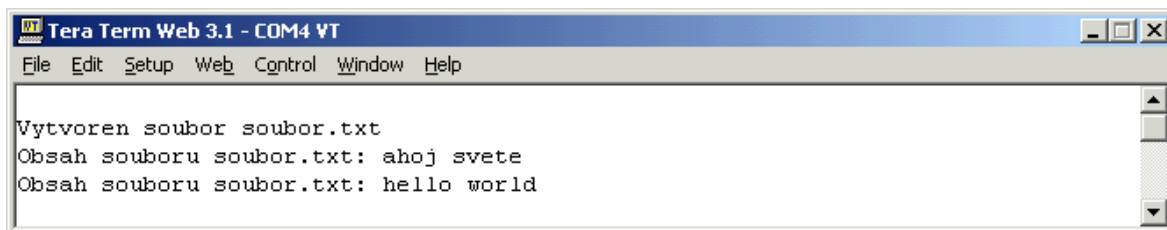
Otevira se soubor soubor.txt pro zapis
Je zapsan retezec hello world

Obsah souboru soubor.txt:
hello world_

```

Obr. 19 Kontrolní výpis po zápisu do souboru

Obrázek Obr. 20 obsahuje zkrácený výpis obsahu souboru po zapsání řetězce hello world



Obr. 20 Výpis obsahu souboru po zapsání řetězce

8.4 Mazání adresáře nebo souboru

Adresář či soubor je smazán zápisem dalšího uzlu dirent s číslem ino = 0. Na obrázku Obr. 21 jsou vidět uzly na flash po smazání adresáře.

00000000:	85 19 02 E0 44 00 00 00	00 00 00 00 01 00 00 00	...00fD0000000000
00000010:	01 00 00 00 00 00 00 00	F4 01 F4 01 00 00 00 00	0000000000000000
00000020:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	0000000000000000
00000030:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	0000000000000000
00000040:	00 00 00 00 85 19 01 E0	30 00 00 00 00 00 00 00	0000...00f0000000
00000050:	01 00 00 00 01 00 00 00	02 00 00 00 00 00 00 00	0000000000000000
00000060:	07 04 00 00 00 00 00 00	00 00 00 00 61 64 72 65	0000000000000000
00000070:	73 61 72 00 85 19 02 E0	44 00 00 00 00 00 00 00	0000000000000000
00000080:	02 00 00 00 01 00 00 00	00 00 00 00 F4 01 F4 01	0000000000000000
00000090:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	0000000000000000
000000A0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	0000000000000000
000000B0:	00 00 00 00 00 00 00 00	85 19 01 E0 30 00 00 00	00000000...00f000
000000C0:	00 00 00 00 01 00 00 00	02 00 00 00 00 00 00 00	0000000000000000
000000D0:	00 00 00 00 07 04 00 00	00 00 00 00 00 00 00 00	0000000000000000
000000E0:	61 64 72 65 73 61 72 00	FF FF FF FF FF FF FF FF	adresar.....
000000F0:	FF FF FF FF FF FF FF FF	FF FF FF FF FF FF FF FF
00000100:	FF FF FF FF FF FF FF FF	FF FF FF FF FF FF FF FF

Začátek uzlu dirent nesmazaného adresáře

Začátek uzlu inode nesmazaného adresáře

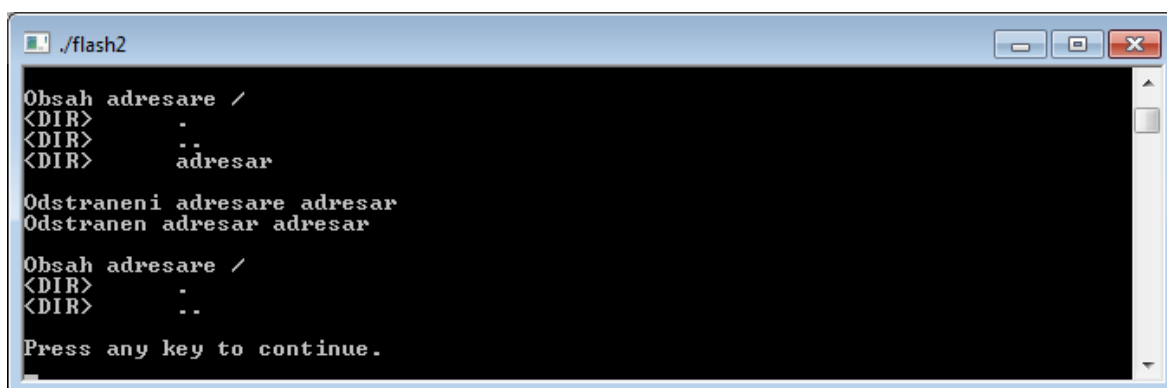
Začátek uzlu dirent, který signalizuje smazaný adresář

Číslo ino nesmazaného adresáře

Číslo ino smazaného adresáře

Obr. 21 Uzly po smazání adresáře

Kontrolní výpis je ukázán na obrázku Obr. 22 kde je vypsán obsah kořenového adresáře před a po smazání.



```
./flash2
Obsah adresare /
<DIR>      .
<DIR>      ..
<DIR>      adresar

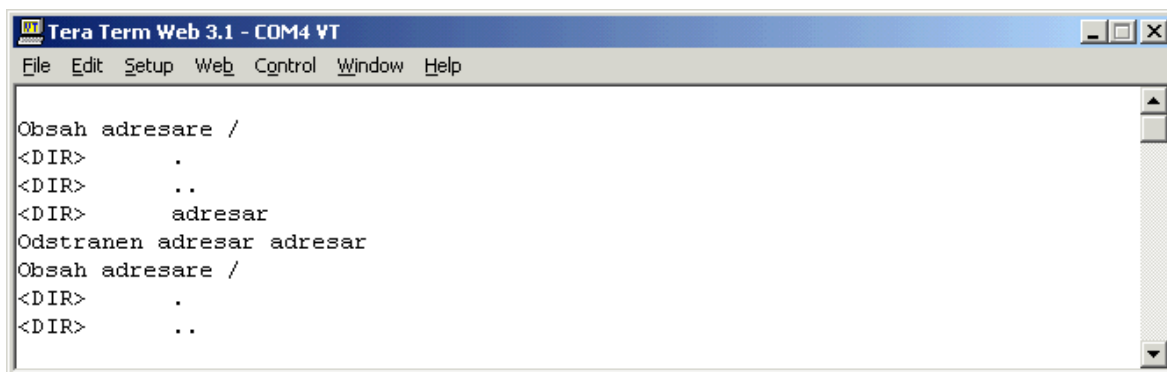
Odstraneni adresare adresar
Odstranen adresar adresar

Obsah adresare /
<DIR>      .
<DIR>      ..

Press any key to continue.
```

Obr. 22 Kontrolní výpis při mazání adresáře

Na obrázku je vidět testovací výpis u provedení funkce smazání adresáře na vývojovém kitu.



```
Tera Term Web 3.1 - COM4 VT
File Edit Setup Web Control Window Help

Obsah adresare /
<DIR>      .
<DIR>      ..
<DIR>      adresar

Odstranen adresar adresar
Obsah adresare /
<DIR>      .
<DIR>      ..
```

Obr. 23 Výpis pro kontrolu správného smazání

9 UKÁZKOVÁ APLIKACE

V rámci diplomové práce byla vytvořena ukázková aplikace, která využívá a demonstuje implementované funkce. Kód vykonávaný aplikací je následující:

- Inicializace flash paměti (viz. podkapitola 6.9).
- Připojení flash paměti (viz. podkapitola 6.2)
- Kontrolní výpis obsahu kořenového adresáře (viz. podkapitola 6.6.4), kořenový adresář je prázdný.
- Vytvoření adresáře s názvem „dir“ v kořenovém adresáři (viz. podkapitola 6.6.3)
- Kontrolní výpis obsahu kořenového adresáře, ve výpisu se zobrazí i položka adresáře „dir.“
- Vytvoření nového souboru s názvem „soubor.txt“ a daty „ahoj svete“ (viz. podkapitola 6.7.5)
- Vstup do adresáře „dir“ (viz. podkapitola 6.6.1), vstup je proveden, aby bylo demonstrováno použití absolutních a relativních cest.
- Otevření souboru „soubor.txt“, protože se tento soubor nachází v kořenovém adresáři je potřeba k němu uvést relativní cestu a to: „../soubor.txt“ otevření souboru je popsáno v podkapitole 6.7.1
- Vypsání dat ze souboru (viz. podkapitola 6.7.4). Soubor obsahuje data „ahoj svete“
- Uzavření souboru (viz. podkapitola 6.7.2)
- Otevření souboru „soubor.txt“, cesta je tentokrát udána absolutně, tedy „./soubor.txt“
- Do souboru je zapsán řetězec „hello world“. Funkce pro zápis je popsána v podkapitole 6.7.3
- Soubor je uzavřen
- Soubor „soubor.txt“ je otevřen, opět je zadána absolutní cesta.
- Je vypsán obsah souboru. Na výpisu se zobrazí text „hello world“
- Soubor je uzavřen

- Přesun o úroveň výše v adresářové struktuře, tedy do kořenového adresáře
- Smazání adresáře „dir“ (viz. podkapitola 6.6.2)
- Kontrolní výpis obsahu kořenového adresáře. Ve výpisu není zobrazen adresář „dir“
- Odpojení flash (viz. podkapitola 6.8).

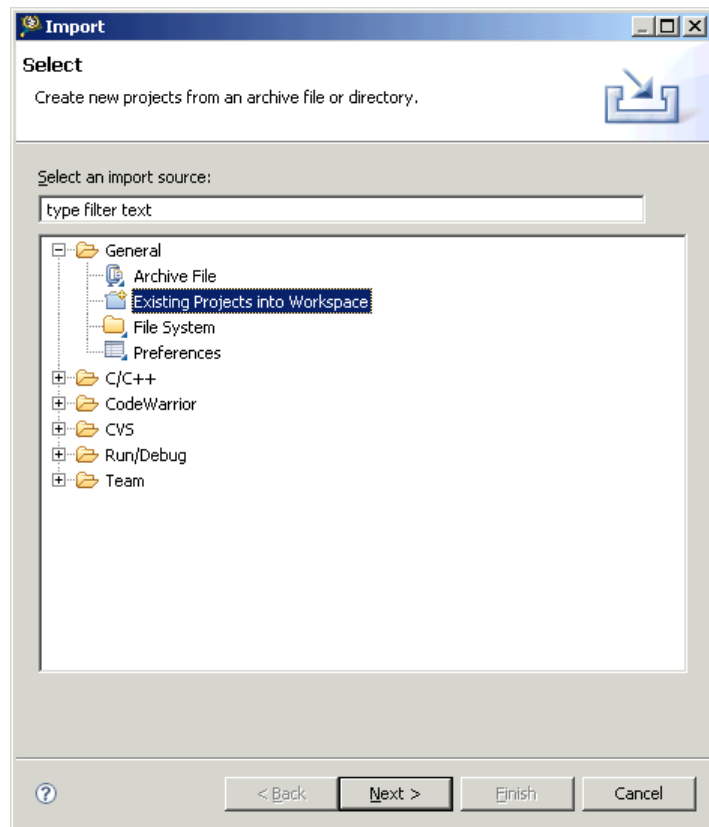
9.1 Návod ke spuštění aplikace na PC

Pro spuštění ukázkové aplikace a její případnou editaci je potřeba vytvořit v některém z vývojových prostředí nový projekt. Při tvorbě diplomové práce bylo použito prostředí CodeLite s překladačem MinGV. Dalším krokem je vložení souborů ze složky jffs2_PC. Také je nutné do složky, ve které vznikne soubor s příponou exe vložit soubor jffs2.img, případně k němu uvést cestu v položce FLASH_NAME. V souboru main.c je zdrojový kód ukázkové aplikace.

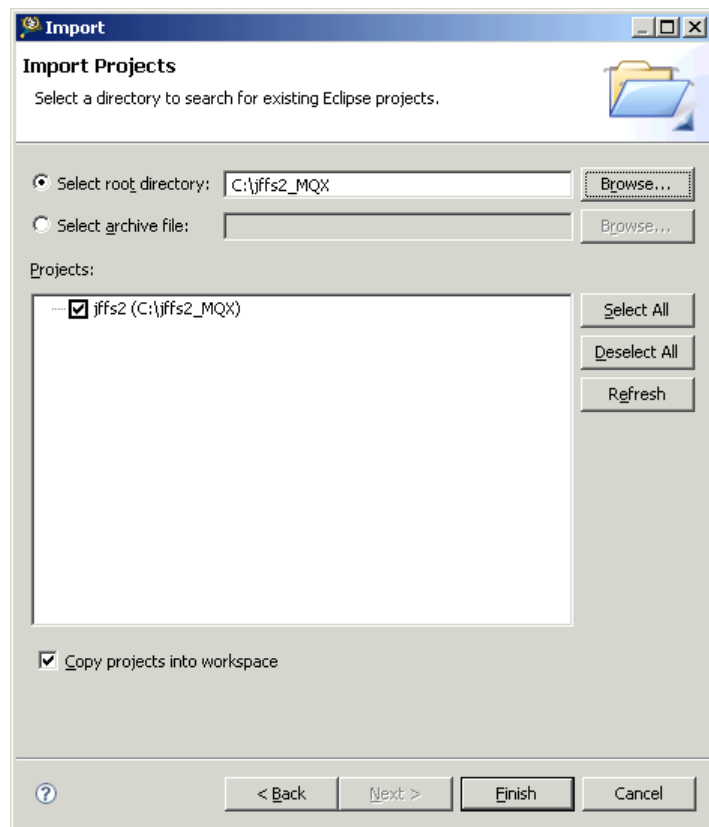
9.2 Návod ke spuštění aplikace na vývojové desce

V případě spuštění ukázkové aplikace a její editace je potřeba se řídit následujícím postupem:

- Po zapnutí vývojového prostředí CodeWarrior 10 zvolit menu File a položku Import.
- V zobrazeném dialogu rozbalit možnost General a zvolit Existing Projects into Workspace viz. Obr. 24
- Po kliknutí na Next vybrat cestu ke složce projektu na CD (složka jffs2_MQX). Cesta bude vypadat například takto D:\jffs2_MQX. Takže se nevybírá soubor projektu, ale je označena celá zdrojová složka projektu viz. Obr. 25
- Zaškrtnout volbu Copy projects into workspace
- Dokončit kliknutím na Finish



Obr. 24 Výběr typu projektu



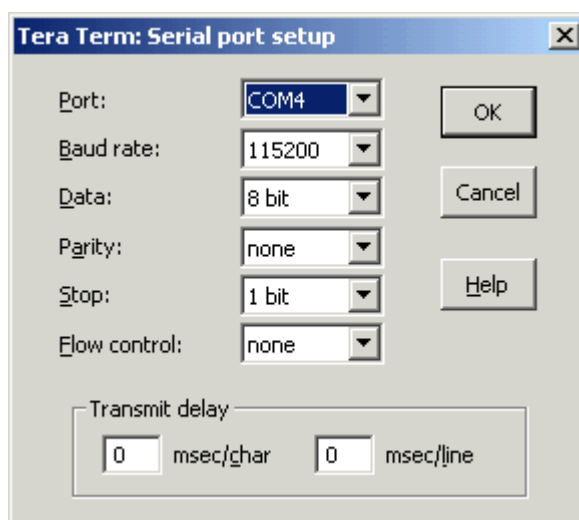
Obr. 25 Zadaní cesty

Po dokončení průvodce se v záložce CodeWarrior Projects zobrazí importovaný projekt.

Zdrojový kód ukázkové aplikace je v souboru main.c

Aplikace je psána pro desku M52277EVB.

Pro zobrazení výpisů aplikace je potřeba propojit PC s konektorem UART0 na vývojové desce přes UBS kabel a případně nainstalovat příslušný ovladač z CD k vývojovému kitu. Pak je potřeba spustit program simulující terminál (např. Tera Term). Dále je nutné nastavit příslušný sériový port na hodnoty dle obrázku Obr. 26



Obr. 26 Ukázka nastavení terminálu

10 SHRNUÍ

Z časových důvodů nebylo možné implementovat všechny části souborového systému JFFS2. Proto jsou v této kapitole shrnuty vytvořené funkce a jsou sepsány návrhy dalšího postupu implementace.

10.1 Připojování

U připojování byly implementovány nejnnutnější postupy pro ukládání uzlů do struktur *jffs2_inode_cache* a *jffs2_raw_node_ref*. To znamená, že při skenování mazatelného bloku, není kompletně vyhodnocováno jeho zaplnění a zastaralost uzlů. Tyto informace využívá garbage collector, který nebyl v rámci práce implementován, a jsou tedy pro funkce vytvořené v rámci diplomové práce nepodstatné. Navíc nebylo možné bez implementace garbage collectoru testovat vyhodnocení obsahu mazatelných bloků.

Dále nejsou ošetřeny stavy, kdy kromě uzlu JFFS2_NODETYPE_CLEANMARKER jsou v mazatelném bloku i další uzly.

Ošetřeny jsou však případné „mezery“ mezi uzly. Tím jsou myšleny případy, kdy další uzel nezačíná na adrese určené z předchozího uzlu, ale je zapsán o několik byte dále.

Také je pro pozdější použití garbage collectoru nutné přidat označování uzlů, které mají být fyzicky smazány. To se děje při kontrole smazaných nebo přejmenovaných adresářů a souborů.

Především je tedy potřeba implementovat zjišťování údajů o stavu mazatelných bloků, označení uzlů k fyzickému odstranění a ošetření případných krizových stavů způsobených nekorektním uložením uzlů.

10.2 Obsah souboru, adresáře

Výpis obsahu souboru či adresáře probíhá bez problému za předpokladu, že jsou uzly zapisovány vždy sekvenčně, a že poslední zapsaný uzel daného souboru nebo adresáře má nejvyšší číslo verze. Takže by bylo vhodné ošetřit i stavy kdy jsou uzly zapsány jiným způsobem, i když by k této situaci docházet nemělo.

10.3 Zápis

Zápisem je myšleno ukládání veškerých nových dat na flash paměť. Takže se jedná nejen o vytváření nových adresářů a souborů, ale i o jejich odstranění. Při mazání je totiž také zapisován nový uzel.

Pokud je v mazatelném bloku dostatek místa, zápis probíhá bez problémů. Pokud zbývá málo místa, je zatím provizorně vybrán další blok v pořadí. Správně by měl být ale vybrán ze seznamu volných bloků. Tento seznam je naplňován při skenování flash paměti. V případě zápisu do souboru a nedostatku místa v mazatelném bloku by měly být vytvořeny 2 uzly. Prvním z nich se vyplní zbývajícím místo ve stávajícím bloku a další bude zapsán do nového bloku. Toto zatím není implementováno, v případě že v aktuálním bloku nezbývá dostatek místa pro zapsání požadovaného množství dat, je zápis proveden do následujícího bloku.

10.4 Návrhy pro postup další implementace

Důležitou částí JFFS2 je garbage collector, který by bylo vhodné implementovat co nejdříve. Postup vykonávání GC je popsán v podkapitole 3.5. Pro jeho správnou činnost je nutné podrobněji zpracovat obsahy jednotlivých bloků paměti flash při připojování média (mount).

Jako další by se měl implementovat výběr vhodného bloku pro další zápis dat. Přehled volných bloků by měl být znám po podrobnějším zpracování obsahu mazatelných bloků.

Toto jsou nezbytné části, bez kterých by nebyl ovladač plně funkční.

Existují ještě další postupy, které by bylo vhodné zařadit. Jedná se především o vyhodnocení nesprávně zapsaných dat. Kontrola validity dat je prováděna metodou CRC. Tato metoda nebyla implementována, ale fyzické uzly obsahují volné místo pro kontrolní součty.

ZÁVĚR

Hlavním cílem této diplomové práce bylo vytvořit ovladač souborového systému JFFS2 pro operační systém Freescale MQX.

V teoretické části byly sepsány základní informace k uvedení do problematiky flash pamětí. Dále byly uvedeny údaje o souborovém systému JFFS2. Taktéž byly shrnuty základní informace o operačním systému MQX a ovladači pro flash paměť v tomto operačním systému.

V praktické části byl jako první popsán způsob komunikace ovladače JFFS2 s flash pamětí a aplikacemi. Dále byly navrženy funkce na rozhraní aplikace – ovladač JFFS2. Pro testovací účely vzniklo simulační prostředí na PC. Výhodou simulace byla vyšší rychlost a neopotřebování flash paměti na vývojovém kitu M52277EVB použitým pro testování.

Při vytváření ovladače byly nejdříve navrženy základní principy důležitých funkcí. Tyto funkce byly následně realizovány a jejich popisy byly rozděleny do několika podkapitol podle účelů funkcí. U vybraných algoritmů byly nakresleny vývojové diagramy.

Funkce na rozhraní ovladač – aplikace prošly testováním na PC i vývojovém kitu. Snímky obrazovky z testování obsahují kontrolní výpisy z terminálu i konzole a také data zapsaná v simulované flash paměti.

Pro názornost vytvořených funkcí byla naprogramována ukázková aplikace, která provádí základní operace s flash pamětí, soubory a adresáři.

Z časových důvodů však nebylo možné implementovat veškerou funkcionalitu JFFS2. Byly vytvořeny funkce pro inicializaci souborového systému na flash paměť, dále připojení a odpojení flash paměti se souborovým systémem. Dalšími funkcemi, které vznikly, jsou funkce pro práci s adresáři a soubory (přidání, odstranění, zápis, čtení). Nebyl však vytvořen garbage collector. Skenování flash paměti neprobíhá v plné míře, jsou získávány pouze nejdůležitější informace pro funkci systému a je potřeba lépe vyřešit výběr dalšího bloku pro zápis dat. Zdrojový kód byl ovšem tvořen tak, aby jej bylo možné vhodně rozšířit a přímo v kódu jsou uvedeny poznámky pro doplňování dalších funkcí.

ZÁVĚR V ANGLIČTINĚ

JFFS2 file-system driver implementation for operating system Freescale MQX was the main point of this thesis.

In the theoretical part basic information about flash memories and the problems related to their usage is outlined. Then the JFFS2 file system is described and also the MQX operating system and its flash memory driver which is used in this work for low-level operations.

In the practical part the proposed scheme of communication of flash memory with the application and JFFS2 driver is described first. Then the design and implementation of the file system is explained. There was also simulation environment created for PC for testing purposes. This environment allows easier and faster testing of the JFFS2 routines during development without wear of the flash memory in real development kit.

First, the basic principles of main functions of the JFFS2 driver were designed. These functions were then implemented. Their descriptions are included in this thesis, divided into several subsections according to the purpose of the functions. Some of algorithms were described by flowchart.

Driver–application interface functions were tested on PC and development kit. Screenshots of testing contain outputs from terminal obtained during testing on real development kit, console outputs obtained from the test application on PC and data in simulated flash memory.

Sample application was created for demonstration of implemented functions. This application performs basic operations with flash memory, files and directories.

All functions of JFFS2 were not implemented because of lack of time. Functions for initialization of the file-system into flash memory, mounting and unmounting flash memory were created. Functions for directories and files operations (creating, deleting, writing, and reading) were created too. Garbage collector was not implemented. Not all information about the file system is obtained from the flash in the current implementation. It is also necessary to develop better way of selecting the next block for writing into file. However, the source code was written in such a way that it should be easy to extend it and directly in the code there are comments, which describe what should be added.

SEZNAM POUŽITÉ LITERATURY

- [1] WOODHOUSE, David. JFFS: The Journalling Flash File System [online]. 2001 [cit. 2011-03-18]. Dostupné z WWW: <<http://sources.redhat.com/jffs2/jffs2.pdf>>.
- [2] ZATLOUKAL, Vít. Flash paměti - fenomén dneška. [online]. 1999, [cit. 2011 - 04-28]. Dostupný z WWW: <http://www.svethardware.cz/art_doc-D5474F7D742D5908C125674200319ADE.html>.
- [3] KASPRZA, Jan. Co umí souborové systémy [online]. 2008 [cit. 2011-03-18]. Dostupné z WWW: <<http://www.fi.muni.cz/~kas/papers/euopen2008-filesystems-proceedings.pdf>>.
- [4] RAATIKAINEN, Kimmo. [Http://www.cs.helsinki.fi/](http://www.cs.helsinki.fi/) [online]. 2007 [cit. 2011-04-28]. Dostupné z WWW: <<http://www.cs.helsinki.fi/group/nodes/Papers/FlashMemories.pdf>>
- [5] Ecos [online]. 2009 [cit. 2011-05-11]. Ecos. Dostupné z WWW: <<http://ecos.sourceforge.org/>>.
- [6] Freescale MQX I/O Drivers Users Guide [online]. [s.l.] : [s.n.], 2011 [cit. 2011-05-11]. Dostupné z WWW: <http://www.freescale.com/files/32bit/doc/user_guide/MQXIOUG.pdf>
- [7] MANN, Burkhard. C pro mikrokontroléry. Praha : BEN - technická literatura, 2004. 280 s. ISBN 80-7300-077-6.
- [8] PINKER, Jiří. Mikroprocesory a Mikropočítače. Praha : BEN ? technická literatura, 2004. 220 s. ISBN 80-7300-110-1.
- [9] CATSOULIS, John. Designing Embedded Hardware. O'Reilly Media, 2005. 400 s. ISBN 978-0-596-00755-3.
- [10] MORTON, Todd D. Embedded Microcontrollers, Prentice Hall, 2001. 694 s. ISBN 0-13-907577.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

DRAM Dynamic Random Access Memory

FAT 32 File Allocation Table (souborový systém)

FTL Flash translation layer.

JFFS Journaling flash file-system

MFS MQX file-system

MQX Operační systém

PC Personal computer

RAM Random Access Memory

SRAM Static Random Access Memory

SEZNAM OBRÁZKŮ

Obr. 1 Fyzický zápis na médium	15
Obr. 2 Hlavička společná pro všechny uzly	16
Obr. 3 Logická struktura uzlů	18
Obr. 4 Postup garbage collectoru	21
Obr. 5 Schéma ovladačů	24
Obr. 6 Algoritmus procházení eraseblocku	33
Obr. 7 Postup části, zpracuj bloky	34
Obr. 8 Postup funkce jffs2_scan_inode_node	36
Obr. 9 Algoritmus funkce jffs2_get_dirent_by_name	39
Obr. 10 Část zpracuj nalezený dirent	40
Obr. 11 Algoritmus funkce získání rodičovského adresáře	42
Obr. 12 Uzel kořenového adresáře	51
Obr. 13 Kontrolní výpis kořenového adresáře	51
Obr. 14 Výpis pro testování na kitu	52
Obr. 15 Uzly na flash médiu po zapsání adresáře	52
Obr. 16 Zapsání adresáře a kontrolní výpis	53
Obr. 17 Výpis u vytvoření adresáře	53
Obr. 18 Uzly po zápisu do souboru	54
Obr. 19 Kontrolní výpis po zápisu do souboru	54
Obr. 20 Výpis obsahu souboru po zapsání řetězce	55
Obr. 21 Uzly po smazání adresáře	55
Obr. 22 Kontrolní výpis při mazání adresáře	56
Obr. 23 Výpis pro kontrolu správného smazání	56
Obr. 24 Výběr typu projektu	59
Obr. 25 Zadaní cesty	59
Obr. 26 Ukázka nastavení terminálu	60

SEZNAM PŘÍLOH

P I Algoritmus výpisu obsahu adresáře

PŘÍLOHA P I: ALGORITMUS VÝPISU OBSAHU ADRESÁŘE

