VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2017 Bc. Vojtěch Vladyka

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

NÁSTROJE PRO DIAGNOSTIKU INTEGRITY SOUBOROVÉHO SYSTÉMU V OS LINUX
DIAGNOSTIC TOOLS FOR OS LINUX FILE SYSTEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Vojtěch Vladyka**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Petr Petyovský, Ph.D.**
SUPERVISOR

BRNO 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Kybernetika, automatizace a měření**
Ústav automatizace a měřicí techniky

Student: Bc. Vojtěch Vladyka ***ID:*** 155259
Ročník: 2 ***Akademický rok:*** 2016/17

NÁZEV TÉMATU:

Nástroje pro diagnostiku integrity souborového systému v OS Linux

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je vytvořit nástroje pro diagnostiku integrity dat souborových systémů v OS Linux, pro které v současnosti tato podpora neexistuje.

1. Seznamte se s problematikou souborových systémů používaných v OS Linux.
2. Zvolte vhodný typ souborového systému (FS), pro který nejsou plně implementovány nástroje pro diagnostiku integrity dat.
3. Definujte možné chyby, které v daném souborovém systému mohou nastat. Definujte chyby, které je možné detekovat a automaticky opravit.
4. Navrhněte a vytvořte vlastní nástroje pro diagnostiku chyb zvoleného souborového systému.
5. Realizujte vlastní nástroje pro opravu chyb v daném souborovém systému (FS).
6. Na vhodných příkladech demonstrujte výsledky diagnostiky a opravy chyb poškozeného FS.
7. Integrujte vytvořené nástroje do prostředí OS Linux. Publikujte tyto nástroje GNU komunitě.
8. Zhodnoťte dosažené výsledky, uveďte výhody a nevýhody řešení a navrhněte další možná rozšíření.

DOPORUČENÁ LITERATURA:

[1] Mitchell, M., Oldham, J., Samuel, A.: Advanced Linux Programming, New Riders Publishing 2001, ISBN 978-0735710436.

[2] TLDP community: The Linux Documentation Project, [HTML dokument]. Dostupný z: < http://www.tldp.org>.

[3] Prata, S.: Mistrovství v C++, Computer press 2004, ISBN 80-251-0098-7.

Termín zadání: 6.2.2017 ***Termín odevzdání:*** 15.5.2017

Vedoucí práce: Ing. Petr Petyovský, Ph.D.
Konzultant:

doc. Ing. Václav Jirsík, CSc.
předseda oborové rady

ABSTRAKT

Tato práce se zabývá problematikou návrhu a implementace nástroje pro detekci a opravu chyb na souborovém systému UDF pro GNU/Linux.

KLÍČOVÁ SLOVA

UDF, fsck, Linux, chyba, CRC, kontrolní součet, GNU, open-source, udffsck

ABSTRACT

Aim of this work is design and implementation of error detection and correction tool for UDF filesystem for GNU/Linux.

KEYWORDS

UDF, fsck, Linux, error, CRC, checksum, GNU, open-source, udffsck

VLADYKA, Vojtěch *Nástroje pro diagnostiku integrity souborového systému v OS Linux*: diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky, 2017. 110 s. Vedoucí práce byl Ing. Petr Petyovský, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Nástroje pro diagnostiku integrity souborového systému v OS Linux“ jsem vypracoval(a) samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor(ka) uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil(a) autorská práva třetích osob, zejména jsem nezasáhl(a) nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom(a) následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora(-ky)

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu semestrální práce panu Ing. Petru Petyovskému Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

podpis autora(-ky)

OBSAH

Úvod **10**

1 Souborové systémy OS Linux **12**
1.1 Fyzická úložná média 12
1.2 Diskové oddíly 13
1.3 Logical Volume Manager (LVM) 13
1.4 Srovnání souborových systémů 15
1.5 Nástroje kontroly integrity 16

2 Universal Disk Format **18**
2.1 Historie UDF 19
2.2 Struktura systému 19
2.3 Deskriptory souborového systému UDF 21
2.3.1 Skupina deskriptorů Volume Recognition Sequence 21
2.3.2 Anchor Volume Descriptor Pointer 22
2.3.3 Skupina deskriptorů Volume Descriptor Sequence 23
2.3.4 Skupina deskriptorů Logical Volume Integrity Sequence 26
2.4 Struktura souborového stromu 27
2.4.1 File Set Descriptor 28
2.4.2 Deskriptor File Entry 28
2.4.3 File Identifier Descriptor 29
2.5 Zhodnocení kapitoly 29

3 Nástroje pro kontrolu konzistence UDF **31**
3.1 Stav projektu udftools 32

4 Definice možných chyb souborového systému **33**
4.1 Kontrola správnosti dat 34
4.1.1 Zabezpečení pomocí kontrolního součtu 35
4.1.2 Zabezpečení pomocí Cyclic Redundancy Check 35
4.2 Použité kontrolní mechanismy v UDF 36

5 Realizace nástroje pro detekci chyb **39**
5.1 Návrh nástroje pro detekci a korekci chyb 39
5.2 Implementace nástroje pro detekci a korekci chyb 41
5.3 Realizace detekce chyb v deskriptoru 42
5.4 Oprava deskriptoru 43
5.5 Mapa chyb 44

5.6 Výstup a ovládání nástroje 45
5.7 Načtení média a jeho příprava na zpracování 46
5.8 Kontrola přítomnosti identifikátoru souborového systému UDF 47
5.9 Načtení, kontrola a korekce deskriptoru AVDP 48
5.10 Načtení, kontrola a korekce deskriptoru VDS 49
5.11 Načtení, kontrola a korekce deskriptoru LVID 50
5.12 Načtení, kontrola a korekce deskriptoru SBD 52
5.13 Kontrola a obnova stromu souborů 52

6 Výsledky a testování nástroje pro detekci a korekci chyb 57
6.1 Testovací data 57
6.2 Metodika testování 57
6.3 Srovnání s ostatními řešeními 59
6.4 Dosažené výsledky 61

7 Publikace nástroje v GNU komunitě 63
7.1 Integrace do balíčku `udftools` 63
7.2 Dopad integrace do balíčku `udftools` 63

8 Závěr 65

Literatura 68

Seznam symbolů, veličin a zkratek 71

Seznam příloh 72

A Výčet operačních systémů podporovaných v souborovém systému UDF 73

B Manuálová stránka k `udffsck` 74

C Ukázkový výpis programu `udffsck` 77

D Výstupy z referenčního řešení `udfct` 80

E Obsah přiloženého DVD 109

SEZNAM OBRÁZKŮ

1.1 Výchozí situace z hlediska rozložení diskových oddílů 13
1.2 Namapování diskových oddílů na LVM 14
1.3 Ukázka přidání nového disku a zvětšení oddílu na LVM 14
2.1 Podpora souborových systémů napříč operačními systémy 18
2.2 Příklad struktury UDF 21
4.1 Schéma kombinace detekčních mechanismů UDF 35
5.1 Zjednodušený algoritmus detekce 40
5.2 Zjednodušený algoritmus detekce a korekce 41
5.3 Zjednodušený algoritmus načtení AVDP 49
5.4 Algoritmus načítání descriptorů FE (funkce *Get File*). Tečkovaně naznačený blok odkazuje na obrázek 5.5. 54
5.5 Algoritmus načítání descriptorů FID (funkce *Inpspect FID*). Tečkovaně naznačený blok odkazuje na obrázek 5.4. 54
6.1 Výsledek kontroly poškozeného média nástrojem `CHKDSK` 58
6.2 Chyba při pokusu o zápis na médium opravené nástrojem `CHKDSK` . . 60

SEZNAM TABULEK

2.1 Struktura VRS deskriptorů 22
2.2 Anchor Volume Descriptor Pointer 23
2.3 Logical Volume Descriptor. Tučně zvýrazněné části jsou kritické pro činnost nástroje pro detekci a korekci chyb. 24
2.4 Partition Descriptor. Tučně zvýrazněné části jsou kritické pro činnost nástroje pro detekci a korekci chyb. 25
2.5 Partition Contents reprezentace. Zvýrazněný identifikátor je jediný, který nástroj bude podporovat. 26
2.6 Space Bitmap Descriptor 26
2.7 Logical Volume Integrity Descriptor. Tučně zvýrazněné části jsou kritické pro činnost nástroje pro detekci a korekci chyb. 27
2.8 File Identifier Descriptor. Tučně zvýrazněné části jsou kritické pro činnost nástroje pro detekci a korekci chyb. 30
2.9 Význam jednotlivých bitů položky File Characteristics deskriptoru FID 2.8 30
4.1 Příklad výpočtu kontrolního součtu 36
4.2 Příklad výpočtu *Cyclic Redundancy Check* s $GP = 1 \cdot x^3 + 0 \cdot x^2 + 1 \cdot x^1 + 0 \cdot x^0$ 37
5.1 Formát struktury `vds_sequence_t` (Mapa chyb) 44
5.2 Formát struktury `metadata_t` 44
5.3 Chybové kódy pro strukturu `metadata_t` 44
6.1 Tabulka detekovatelných a opravitelných chyb 62
A.1 Seznam oficiálně podporovaných OS 73

ÚVOD

Diagnostika integrity dat je klíčovou částí správy a uchovávání dat. Bez ní nemáme žádnou informaci o jejich konzistenci ani o jejich správnosti a vzhledem k faktu, že data jsou nejcennější část výpočetní techniky, je potřeba této problematice věnovat dostatečnou pozornost.

Byla to právě data co stálo na začátku zrodu výpočetní techniky. Bylo jich stále více a lidské zdroje přestávaly na narůstající objem stačit. Původní úložné médium byla papírová karta ve složce v kartotéce a ta byla seřazena podle určitého řádu. Za ekvivalent dnešních metadat souborového systému by bylo možné považovat popisné štítky složek a zásuvek a rešeršní katalogy. Pokud došlo k chybě při založení složky, šance, že na to někdo přijde dřív, než ji bude potřebovat byla prakticky nulová. Jinými slovy, diagnostika a kontrola integrity dat neexistovala, předpokládalo se, že systém je natolik snadný, že není potřeba. Ovšem ve chvíli, kdy psaný text nahradil kód, nejprve ve formě otvorů na páskách a štítcích a později magnetické záznamy, člověk už nedokázal případnou zatoulanou složku ani poznat, natož opravit její zařazení. A tady začíná pole působnosti nástrojů kontrolující integritu dat.

Cílem této práce je vytvořit nástroj pro diagnostiku konzistence dat souborového systému v GNU/Linux, pro který tyto nástroje neexistují. Bude provedena rešerše existujících souborových systémů a jejich diagnostických nástrojů a vybrán dostatečně perspektivní souborový systém, kterému tyto nástroje chybí. Dostatečně perspektivním je myšlen souborový systém, který má i v dnešní době využití, ať už z historických důvodů nebo z důvodu popularity. Perspektivní by rozhodně nebylo dopisovat chybějící nástroje pro souborový systém nasazený například pro historický operačná systém *Unix System V*.

Různé souborové systémy jsou určeny přednostně pro určitá fyzická média. Tomu odpovídá i jejich design, který obvykle respektuje silné a slabé stránky daného média. To také odráží i druh chyb, které se na daném médiu vyskytují a to, jak se s nimi vypořádává souborový systém. Lze předpokládat, že jinými ochrannými mechanismy bude opatřeno optické médium, které je samo vystaveno kontaktu s fyzickým světem a jinými pevný disk, který je bezpečně skrytý v kovovém těle. Stejně tak se objeví jiné chyby u sekvenčně zapisovaných médií a u médií s náhodným přístupem.

Obecně lze tvrdit, že jsou chyby detekovatené a nedetekovatelné. Detekovatelnost je otázkou použitého detekčního mechanismu, který má určitou sílu a pokud ji chyba převýší, stane se nedetekovatelnou. Snahou tvůrců souborových systémů je pokrytí co největšího množství chyb alespoň jejich detekcí aby bylo možné chráněná data prohlásit za neplatná.

V doplňku k tomu stojí opravitelnost chyb. Stejně jako v předchozím případě opravitelnost stojí na síle použitého algoritmu. A i opravitelná chyba se může stát

neopravitelnou, pokud jejich množství překoná schopnosti opravného algoritmu. To, které chyby opravit lze či ne ovšem není možné obecně říct bez znalosti použitých opravných algoritmů. Proto je potřeba pečlivě nadefinovat jaké chyby mohou nastat na daném souborovém systému a mohou být detekovány a kolik z nich je ještě možné opravit pomocí nástrojů poskytnutých tvůrci souborového systému.

Ve druhé části práce je popsáno vytvoření nástroje pro vybraný souborový systém *Universal Disk Format*. Je popsána problematika jeho návrhu a následné implementace detekce a opravy jednotlivých definovaných poruch média. K tomuto patří i popsání způsobu otestování výsledného řešení na vhodně zvolených příkladech.

Závěr práce je věnován začlenění vziknuvšího nástroje do komunity GNU a dopad tohoto kroku na GNU/Linux.

1 SOUBOROVÉ SYSTÉMY OS LINUX

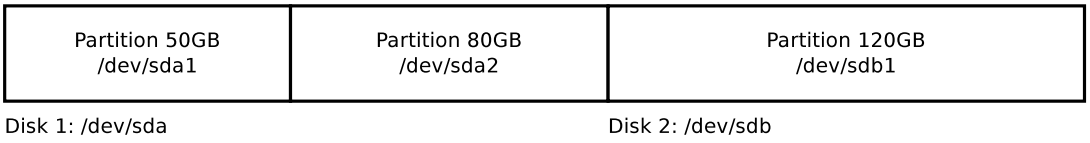
1.1 Fyzická úložná média

Fyzická úložná média představují nejnižší vrstvu a zároveň základní kámen všech souborových systémů, protože právě na základě jejich vlastností byly vytvářeny.

Z historického hlediska po současnost vývoj přešel přes několik technologií počínaje děrnou páskou a děrnými štítky, přes magnetické ukládání informace do různých nosných médií, optická média po flash paměti. Z hlediska souborových systémů má smysl mluvit až o magnetických médiích, která byla použita pro běžnou práci, protože například lineární pásková úložiště nebyla konstruována pro kompatibilitu s ostatními systémy ale pouze pro vlastní potřeby, tudíž záznam neobsahoval metadata, ale pouze data samotná a jejich struktura byla uložena jinde nebo vůbec. První opravdové použití souborového systému přišlo až s nástupem disket a pevných disků.

První opravdový souborový systém v Unixu byl **S5FS** neboli *System V File System* (1974), často ovšem označovaný pouze jako **FS** (File System) nebo **UFS** (Unix File System), který byl implementován v *UNIX System V* běžící například na mainframech PDP-11. Tento souborový systém byl velice pomalý a také poměrně jednoduchý, ale právě jeho architektura posloužila jako vzor většiny následujících unixových souborových systémů. Tomu ovšem předcházel souborový systém pro mikropočítače s názvem **CP/M** (1973) neboli *Control Program/Monitor* a později *Control Program for Microcomputer*. Tento souborový systém byl určen narozdíl od S5FS pro mikropočítače od společnosti Intel. V době vzniku totiž začal Intel vyrábět první osmibitové procesory Intel 8008 a o rok později v roce 1974 vznikl Intel 8080 a právě s ním se rozšířil i CP/M. CP/M byl někde na pomezí mezi souborovým a operačním systémem. Na úložném médiu (v tomto případě disketě) byl BIOS, který poskytoval systémová volání na zápis a čtení, řízení klávesnice a monitoru. Poté bylo CP/M, které využívalo těchto volání a poskytovalo je spolu s dalšími shellu a uživatelskému programu. Systémová volání, která přidalo CP/M se týkaly převážně přístupu k souborům, které CP/M uchovávalo. Dalším významným souborovým systémem, který vznikl pro diskety, byl **FAT12** (1977). Posléze s rozšířením pevných disků se začaly vyvíjet další nové souborové systémy odpovídající požadavkům.

Z hlediska struktury disku lze mluvit o CHS (*Cylinder-Head-Sector*), které rozdělilo úložné médium ve třech rozměrech podle jednotlivých stop (*Cylinder*), čtecí hlavy (*Head*), kterých může být u pevných disků více, záleží na počtu jednotlivých disků a použitých stran a poslední rozměr je sektor (*Sector*), který určuje, o kterou část stopy jde. Právě sektory dostály ve vývoji největších změn, protože původní dě-



Obr. 1.1: Výchozí situace z hlediska rozložení diskových oddílů

lení po úhlech od středu disku nebylo efektivní se vzrůstající vzdáleností od středu. Proto se později přešlo k dělení na jednotnou velikost sektoru, které se rozložily rovnoměrně po celém disku.

Sektor je nejmenší zápisovou jednotkou na médiu, která je určená výrobcem média. Typicky se jedná o velikosti 512 B u pevných disků, 2048 B u CD-ROM a DVD-ROM. Nové pevné disky používají velikost bloku 4096 B. Důvodem pro členění po sektorech je zjednodušení přístupu k médiu za předpokladu, že uložená data jsou větší než velikost bloku. V opačném případě dochází k plýtvání místem, protože se vždy čte a zapisuje celý blok. Právě proti sektorům se obvykle navrhují bloky souborového systému, které se snaží využít sektory co nejefektivněji jak z hlediska úspory místa, tak z hlediska přístupového a zápisového času.

1.2 Diskové oddíly

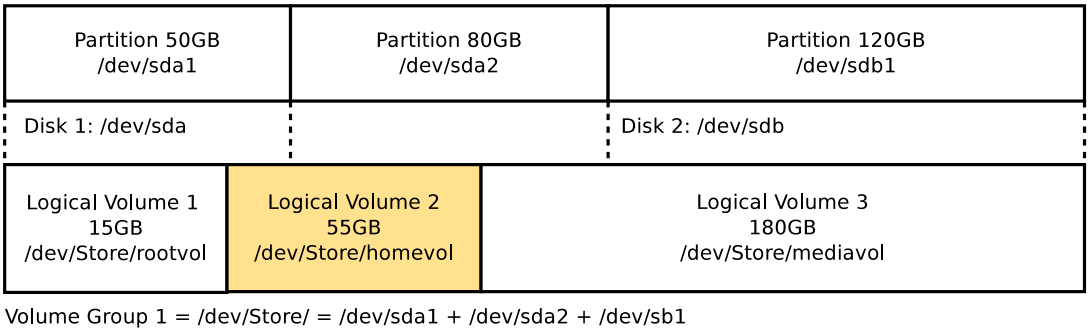
Diskové oddíly představují způsob, jak lze fyzické médium abstraktně rozdělit na více částí. Rozdělení na oddíly je uloženo v MBR (Master Boot Record) a jeden disk může obsahovat až 4 hlavní oddíly. V případě potřeby více oddílů lze použít rozšířený oddíl, který zapouzdřuje další dělení na oddíly, ovšem tentokrát již mimo MBR.

Důvodem pro rozdělení na oddíly může být oddělení uživatelských dat od systémových, vyhrazení místa pro odkládací oddíl nebo jiné logické dělení.

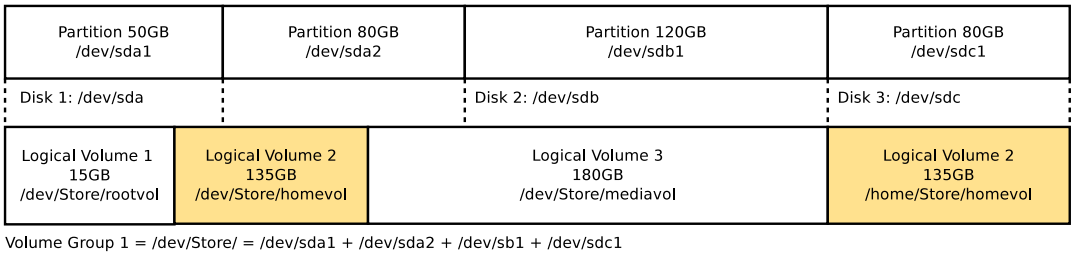
1.3 Logical Volume Manager (LVM)

Logical Volume Manager (LVM) představuje způsob abstrakce nad diskovými oddíly. Ačkoli se může zdát, že se jedná o podobnou technologii jako je RAID, není tomu tak (více v [15]).

LVM vzniklo pro zajištění větší flexibility pro organizaci úložiště. Představte si situaci na obrázku 1.1. Jedná se o skupinu dvou pevných disků (`/dev/sda` a `/dev/sdb`), které jsou z nějakých důvodů, které nejsou podstatné, rozděleny na zobrazené oddíly.



Obr. 1.2: Namapování diskových oddílů na LVM



Obr. 1.3: Ukázka přidání nového disku a zvětšení oddílu na LVM

Naším cílem je vytvořit tři diskové oddíly s velikostmi 15 GB, 55 GB a 180 GB. Možnosti jsou dvě. První je RAID 0 a druhou je LVM. Pokud se rozhodneme pro řešení RAID 0, sloučíme disky dohromady do pole a naformátujeme je na požadované oddíly. Totéž můžeme vyrobit i pomocí LVM, jak je naznačeno na obrázku 1.2. Řešení pomocí RAID 0 by vypadalo podobně.

Pokud se nyní soustředíme trochu více na LVM, můžeme si všimnout, že všechny oddíly byly sloučeny do jednoho logického oddílu `/dev/Store` s velikostí 250 GB. Tento byl posléze logicky rozdělen na jednotlivé oddíly. Ty mohou poté být naformátovány libovolným souborovým systémem.

Zásadní rozdíl oproti RAID 0 je v možnostech změny struktury úložiště. V případě RAID 0 se změna promítá do formátování RAID úložiště, v případě LVM se jedná o změnu velikosti logické jednotky pouze za předpokladu existence nějakého volného místa, nezávisle na jeho fyzickém umístění na disku. Tudíž není třeba přesouvat oddíly pro potřeby jejich zvětšení. Toto je demonstrováno na obrázku 1.3.

V situaci na obrázku došlo k přidání nového disku `/dev/sdc` do logického oddílu `/dev/Store` a přiřazení nově vzniklého místa již existujícímu oddílu. Povšimněte si oranžově zvýrazněného oddílu `/dev/Store/homevol` na obrázku 1.2 a nyní na obrázku 1.3. Jeho umístění první části zůstalo zachováno a nové místo se pouze připojilo. Stejně tak oddíl `/dev/Store/mediavol` nebylo třeba přesouvat. Jediný zásah, který je nutný pro používání nového místa, je na straně soubového systému,

t.j. jeho zvětšení na celý logický oddíl.

Samozřejmě není nutnost veškeré místo přiřadit pouze jedinému oddílu, místo můžeme dle libosti přiřazovat oddílům, či vytvářet oddíly nové.

Nutno podotknout, že toto řešení nenahrazuje RAID. Ten má stále svou sílu ve svých hardwarových řadičích, tudíž jeho výkonnost je vyšší, ale to je daň za jeho nižší flexibilitu.

1.4 Srovnání souborových systémů

Pro operační systém Linux bylo během jeho existence vyvinuto mnoho různých souborových systémů, stejně tak jako jich bylo mnoho implementováno z ostatních operačních systémů. Některé z nich jsou používány i v současnosti, některé byly opuštěny nebo nahrazeny.

Podle [16] patří mezi nejvýznamnější nativní souborové systémy tyto:

- **ufs** – původní souborový systém, který vznikl pro *Unix V*. Posloužil jako vzor pro další vývoj ostatních souborových systémů pro unixové operační systémy. Zavádí pojem *Inode*, který posléze přebírají souborové systémy **ext**.
- **minix** – nejstarší souborový systém OS Linux. Vzhledem k jeho stáří se dnes již aktivně nevyvíjí a nepoužívá. Mezi jeho hlavní omezení patří maximální velikost souborového systému, která je až 64 MB. Dalším významným omezením je délka názvů souborů, která je limitována na 30 znaků.
- **ext** – tento souborový systém, celým jménem Extended Filesystem, vznikl v roce 1992 speciálně pro potřeby kernelu OS Linux [18]. Jeho inspirací byl souborový systém UFS (Unix File System), ze kterého převzal strukturu metadat. Jeho limitací byla maximální velikost 2 GB.
- **ext2** – nástupce souborového systému ext. Byla navýšena maximální velikost na 32 TB a byl navržen podle stejných principů jako Berkeley Fast File System z projektu BSD [17]. Dlouhou dobu byl tento souborový systém používán ve všech hlavních distribucích OS Linux. ext2 nebyl zpětně kompatibilní s původním ext.
- **ext3** – již v pořadí třetí verze ext souborového systému. Nyní již byla zachována téměř plná zpětná kompatibilita s ext2. Oproti ext2 ovšem ext3 přineslo žurnálování, které pomohlo v obnově souborového systému při chybě systému [19]. Zbytek parametrů zůstal shodný s ext2.
- **ext4** – současná verze ext souborového systému [20]. Opět je zpětně kompatibilní s ext3 a ext2, ale oproti nim byla navýšena maximální velikost na 1 EB, ale doporučené maximum je 16 TB. Původně byl ext4 pouze skupina rozšíření pro ext3, ale později se osamostatnil jako ext4. Dalším rozšířením je

nyní již neomezený počet podsložek (ext3 mělo omezení na 32000 podsložek). Také bylo zlepšena odolnost žurnálu přidáním kontrolních součtů a celková výkonnost souborového systému.

- **vfat** – typický zástupce cizích souborových systémů. V tomto případě se jedná o MS FAT32, který je hojně používaný na USB flash discích.
- **iso9660** – standardní souborový systém pro CD-ROM.
- **udf** – souborový systém původně vyvinutý pro datové rozšíření CD-ROM. Posléze rozšířen pro DVD a BluRay. V současnosti je používán například v šifrovaných flash discích.
- **smbfs** – síťový souborový systém Samba File Systém vyvinutý pro sdílení dat mezi počítači s MS Windows. Podporuje Windows Sharing Protocol.
- **ntfs** – *New Technology File System* je v současnosti nejpoužívanější souborový systém v MS Windows. Jedná se o proprietární souborový systém, který má nativní podporu pouze pro MS Windows. Pro OS Linux vznikla open-source varianta ovladače, která umožňuje jak čtení, tak zápis na **ntfs**. MacOS má zabudovanou pouze omezenou podporu čtení a zápisu, ale zápis je ve výchozím nastavení zakázán. Zde je možné ovladač dokoupit od třetí strany [34].
- **btrfs** – nový souborový systém založený na principu *copy-on-write* (COW) pro OS Linux. Umonžňuje změnu velikosti souborového systému za jeho běhu, stejně tak přidávání fyzických úložišť. Podporuje kontrolní součty nad daty.
- **hfs** – proprietární souborový systém vyvinutý společností Apple jako náhrada původního *Macintosh File System*, který vznikl pro diskety a na pevné disky již výkonově nestačil.
- **hfs+** – náhrada souborového systému **hfs**, který slouží primárně pro Apple macOS. Opět je proprietární a oproti svému předchůdci podporuje například velké soubory (až 8 EB).

1.5 Nástroje kontroly integrity

V prostředí OS Linux je pro kontrolu integrity určen primárně nástroj `fsck` (Filesystem Consistency Check). V manuálové stránce k `fsck` [21] se lze dočíst, že samotný `fsck` je pouze frontend, který volá specifické nástroje pro daný souborový systém. Skutečné nástroje určené pro kontrolu integrity, které jsou obvykle dodávány spolu s nástroji k vytvoření a práci s daným soborovým systémem, se jmenují `fsck.fstype`, například `fsck.ext4`.

Pro masivně používané souborové systémy tyto nástroje existují. Pro **ext** rodinu se jedná o nástroj `e2fsck`. Podporovány jsou i převzaté souborové systémy, například `fsck.fat` pro souborové systémy **vfat** a **msdos**. Ovšem například pro **iso9660**

nástroje kontrolující integritu existují pouze jako nástroje vzniklé na základě pokusů při testování vadných disků. Takovým je i `isovfy` [33] ze skupiny nástrojů isoinfo, který ovšem umožňuje pouze kontrolu integrity, nikoli opravu chyb.

Nástrojem, který v tuto chvíli chybí, je podle všeho `fsck.udf`. Projekt s názvem udftools [8], byl autorem opuštěn v roce 2004 a nástroj pro kontrolu konzistence nebyl nikdy započat a proto se v dalších částech práce soustředím právě na tento souborový systém.

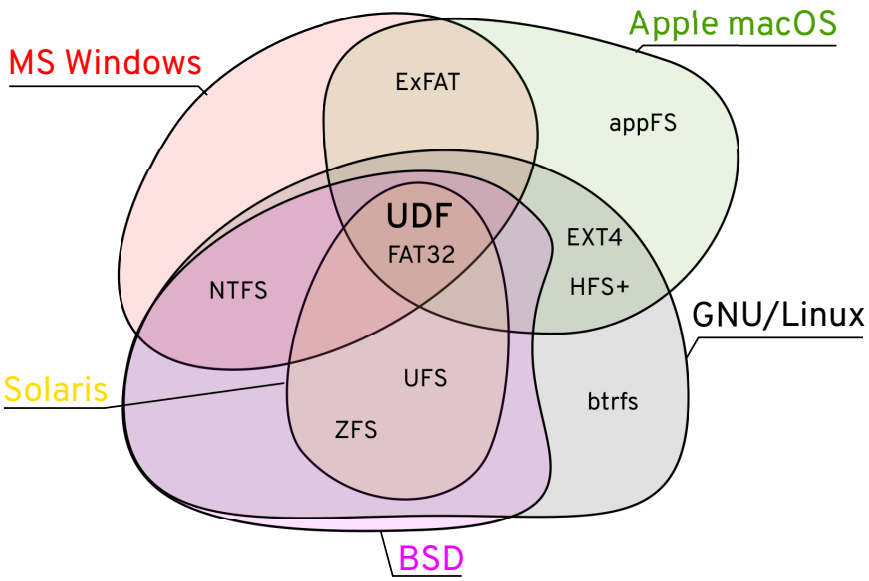
2 UNIVERSAL DISK FORMAT

Universal Disk Format (UDF) je souborový systém navržený pro použití na CD, DVD a Blu-Ray discích, ale díky jeho univerzálnosti je možné jej použít i na jiných médiích. Jeho návrh vyšel ze systému CDFS (ISO 9660). S ním je kompatibilní pouze na úrovni možné koexistence na stejném médiu a ve formátu VRS.

Mezi základní vlastnosti UDF patří:

- Podpora napříč všemi používanými OS (kompletní výčet je v příloze A).
- Podpora specifických metadat jednotlivých OS na úrovni souborového systému (například *Mac Finder Info*).
- Věkem ověřený souborový systém (první verze UDF vznikla v roce 1995, poslední vydaná verze je z roku 2005).
- Postavené na mezinárodním standardu ECMA-167/ISO 13346 [4]. Výsledná specifikace je volně dostupná (specifikace standardu UDF 2.01 [2]).
- Robustní struktura metadat, kritická metadata jsou uložena rednundantně.
- Velikost oddílu 2^{32} bloků (2 TB pro 512B velikost bloku).
- Jména souborů mohou mít délku až 254 B a jsou uložena v Unicode.
- Podporuje symbolické odkazy (symlink)

Na obrázku 2.1 je zobrazena podpora vybraných souborových systémů ve vybraných operačních systémech. Jak je vidět, pouze UDF a FAT32 jsou jediné, které jsou podporované napříč všemi operačními systémy. Důvodem, proč je UDF nadřazené FAT32, je robustnost celého souborového systému, která je daná návrhem standardu UDF.



Obr. 2.1: Podpora souborových systémů napříč operačními systémy

2.1 Historie UDF

První verze nového souborového systému *Universal Disk Format* vznikla v roce 1995 ve společenství firem s názvem *Optical Storage Technology Association* (OSTA). Důvody pro vznik nového souborového systému byly hlavně zvyšující se nároky na práci s optickými médii, kde souborový systém ISO 9660 přestával stačit. Jmenovitě se jednalo o blížící se příchod DVD a posléze CD-RW.

Historie verzí UDF v chronologickém pořadí je zachycené v následující seznamu. Jak je vidět, číslování není postupné, toto je kompletní výčet. Mezi jednotlivými verzemi vycházely tzv. DCN dokumenty. Dokumenty s čísly 5100 až po 5122 se týkají UDF standardu a popisují nalezené problémy v jednotlivých verzích UDF a jejich případná řešení. Změny a opravy standardu UDF byly zanášeny na základě těchto dokumentů. Poslední verze je dostupná online [35].

- verze 1.00 (24. října 1995) - první vydaná verze
- verze 1.01 (3. listopadu 1995) - opravy chyb z verze 1.00, přidán dovětek týkající se použití na DVD-ROM
- verze 1.02 (30. srpen 1996) - další opravy verze 1.01, začlěnení DVD-ROM a DVD-Video do specifikace, tato verze byla původně použita pro DVD-Video
- verze 1.50 (4. února 1997) - přidána podpora pro CD-R a CD-RW (podpora VAT, správa chyb)
- verze 2.00 (3. dubna 1998) - začlenění změn z revize 3 standardu ECMA-167 [4], přidání podpory tzv. *Named Streams*, změna fungování VAT
- verze 2.01 (15. března 2000) - opravy chyb verze 2.00
- verze 2.50 (30. dubna 2003) - přidání *Metadata partition*, navrženo pro HD-DVD a Blu-Ray
- verze 2.60 (26. ledna 2005) - přidání pseudo-přepisu pro sekvenční média
- verze 2.60 (1. března 2005) - redakční opravy

Jak je vidět u verze 2.00, UDF nevzniklo z ničeho, ale bylo vybudováno nad standardem ECMA-167 [4], který popisuje jednotlivé deskriptory systému a jejich navrhované použití. UDF přidává svou funkcionalitu do částí popsaných jako *Contents Use*, případně *Application Use*, což jsou části vyhrazené pro konkretizaci použití standardu ECMA-167. V době psaní prvního standardu UDF existovala ECMA-167, revize 2. S revizí 3 vzniklo UDF 2.00 a od té doby nová revize nevyšla.

2.2 Struktura systému

Struktura UDF je naznačena na obrázku 2.2. Jak je vidět, obsahuje základní skupiny deskriptorů (jejich popis je v kapitole 2.3) a data. Nutno podotknout, že čísla sektorů jsou pouze ukázková; jediná, která jsou platná jsou sektory 16 (umístění

VRS), 256 (umístění AVDP) a poslední sektor (umístění AVDP). Umístění ostatních deskriptorů je závislé na dané implementaci, která vytváří souborový systém.

Norma popisující UDF definuje dva druhy číslování. Prvním je *Logical Sector Number* (LSN), které popisuje čísla jednotlivých sektorů. Je číslované od nuly a začíná na prvním fyzickém sektoru média. Jeho velikost odpovídá fyzické velikosti sektoru média. Jeho cílem je fyzická navigace na médiu.

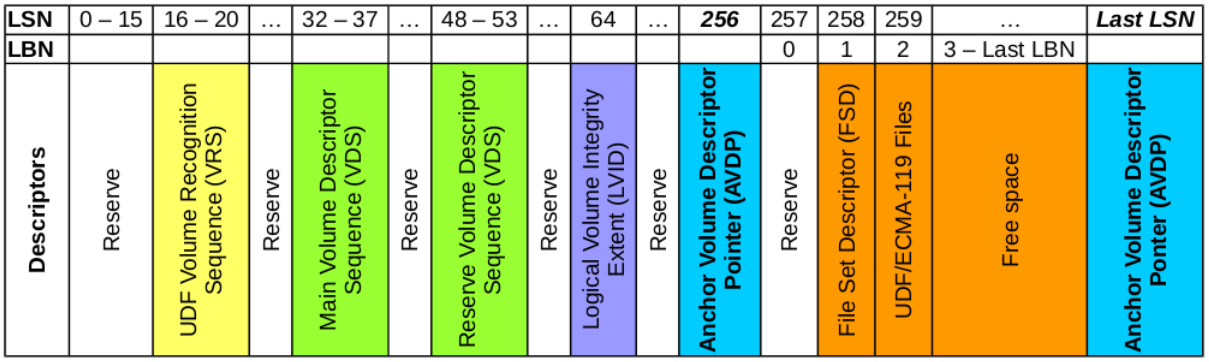
Druhým číslováním je *Logical Block Number* (LBN), které popisuje čísla jednotlivých logických bloků, které jsou mapovány na sektory. Jejich velikost by měla být stejná jako velikost LSN (pokud není, je to v rozporu se standardem UDF). Je číslované od nuly ale začíná až v oblasti dat na začátku logického svazku. Jeho cílem je na rozdíl od LSN navigace pouze v rámci logických svazků.

UDF nese tři příznaky ukazující na jeho stav z hlediska práce s ním. Jedná se o *Access Type*, *Sequentiality* a *Finalization*.

Access Type značí možnosti zápisu na médium. Je to údaj uložený v PD (kapitola 2.3.3.) Může být *Read Only* (např. CD-ROM), *Write Once* (např. CR-R, DVD-R), *Rewritable* (např. CR-RW) nebo *Overwritable* (např. DVD-RAM). Tuto informaci musí dodat řadič média. Tento údaj je důležitý pro způsob zápisu na médium. *Read Only* média není třeba řešit, protože zápis nikdy neproběhne. *Write Once* média závisí na stavu *Finalization* - pokud je médium uzavřeno, není možné nic dopsat. Pokud je stále otevřeno, je možné data doplnit, případě mazat (obsazené místo je ale nenávratně obsazené). *Rewritable* média je možné mazat, ale vzhledem k velice omezenému počtu zápisových cyklů není žádoucí zapisovat co není potřeba. Proto se indikuje tento údaj pro omezení zápisů na médium na nutné minimum. To je ostatně také jeden z hlavních důvodů existence *Space Bitmap Descriptor*, aby bylo naznačeno, které bloky je potřeba před dalším zápisem přemazat a které jsou prázdné. Poslední možností je *Overwritable*, tato možnost je určena pro média s „neomezeným" zápisem na médium (neomezeným v porovnáním s CD-RW, například DVD-RAM nebo Flash média.)

Sequentiality popisuje, zda je médium sekvenční (například CD-R) nebo nesekvenční (*Random Access*, například Flash disk). Pro sekvenční média totiž UDF poskytuje *Virtal Allocation Table*, která vytváří dojem nesekvenčního (*Random Access*) média.

Finalization popisuje, zda bylo vytváření média ukončeno. Jedná se o binární stav, médium je buď ukončeno (*Closed*, případně *Finalized* nebo otevřeno *Open*). Tento stav je odvozen od počtu AVDP uložených na médiu. Pokud je uloženo pouze jediné (na sektoru 256 nebo 512), médium je otevřené (*Open*) a struktura jeho metadat může být ještě měněna v rámci zápisu. Pokud je uloženo AVDP vícekrát (alespoň dvakrát a to na sektorech 256 a buď posledním sektoru (N) nebo na sektoru $N - 256$), médium je považované za dokončené (*Finalized*).



Obr. 2.2: Příklad struktury UDF

2.3 Deskriptory souborového systému UDF

Deskriptory souborového systému, neboli metadata souborového systému, jsou klíčovou částí každého FS. UDF používá pět skupin deskriptorů pro svou funkci. Jedná se o tyto:

- *Volume Recognition Sequence* (VRS) - oblast definující přítomnost UDF 2.3.1.
- *Anchor Volume Descriptor Pointer* (AVDP) - deskriptory ukazující na VDS, vstupní bod do UDF 2.3.2.
- *Volume Descriptor Sequence* (VDS) - hlavní skupina deskriptorů popisující UDF. Na souborovém systému je uložena dvakrát 2.3.3.
- *Logical Volume Integrity Descriptor* (LVID) - deskriptor (případně skupina deskriptorů) popisující integritu souborového systému 2.3.4.
- Deskriptory oblasti dat 2.4

Vzhledem k velkému překryvu mnou popsaných informací s informacemi, které jsou uvedeny ve standardu UDF 2.01 [2] a jeho nadřazeném standardu ECMA-167, revize 3 [4] je tato část pojata pouze jako přehledová a neklade si za cíl absolutně popsat UDF. Cílem je pouze informativně vyzdvihnout důležité deskriptory UDF.

2.3.1 Skupina deskriptorů Volume Recognition Sequence

Volume Recognition Sequence (VRS) je sekce určená k rozpoznání obsaženého souborového systému. Může být velká až 6 sektorů a obsahuje deskriptory identifikující souborový systém. Jeho poloha je fixní na sektoru 16.

Důvodem, proč je poloha VRS na sektoru 16, je zpětná kompatibilita a možná koexistence se souborovým systémem ISO 9660. ISO 9660 používá tuto oblast také jako VRS, tudíž je pochopitelné, že ji UDF převzalo. Nese to s sebou i některé výjimky právě kvůli kompatibilitě, například co se týká velikosti sektoru, která je

Adresa	Délka [B]	Jméno položky	Datový typ	Popis
0	1	Type	int8	Typ deskriptoru
1	5	Identifier	string	Identifikátor typu souborového systému
6	1	Version	int8	Verze deskriptoru
7	2041	—	—	Volné místo, záleží na typu deskriptoru

Tab. 2.1: Struktura VRS deskriptorů

použita v této oblasti. Problematika načítání a zpracování VRS je popsána v kapitole 5.8.

Struktura deskriptorů použitých ve VRS je zachycena v tabulce 2.1.

Jak je vidět, deskriptor se skládá ze třech částí, pomineme-li poslední rezervovanou část obsahující volné místo. První položka *Type* určuje typ deskriptoru. V tomto případě je jediná povolená hodnota 1. Další položka s názvem *Identifier* obsahuje řetězec o pěti znacích. Tato část určuje typ použitého souborového systému. Možné varianty jsou tyto:

- `BOOT0` - Bootovací záznam
- `CD001` - Souborový systém ISO 9660
- `CDW02` - Souborový systém podle ECMA-168 [5]
- `BEA01` - *Beginning Extended Area Descriptor*, začátek UDF Bridge sekvence
- `TEA01` - *Terminating Extended Area Descriptor*, konec UDF Bridge sekvence
- `NSR01` - UDF verze 1.00 a vyšší
- `NSR02` - UDF verze 1.50 a vyšší
- `NSR03` - UDF verze 2.01 a vyšší

Poslední položkou je *Version* která obsahuje informaci o verzi deskriptoru. Zde by měla být v těchto případech vždy `0x01`.

Aby mohlo být prohlášeno, že médium obsahuje UDF, musí být nalezen deskriptor obsahující *Identifier* s hodnotou `NSR01`, `NSR02` nebo `NSR03`. V opačném případě se nejedná o UDF.

2.3.2 Anchor Volume Descriptor Pointer

Anchor Volume Descriptor Pointer (AVDP) je klíčová část souborového systému. Tato struktura udržuje adresu VDS a jeho délku a to jak pro hlavní, tak pro záložní VDS. Právě AVDP je první deskriptor, který se čte po identifikaci souborového systému, proto je umístěn na předem známém místě. V případě uzavřených souborových systémů (více o *Finalization* v kapiole 2.2) to je na sektoru 256, posledním sektoru média a nezřídka také na 256. sektoru od konce média. Podmínkou je, že AVDP musí být uloženo alespoň dvakrát. V případě otevřených souborových systémů je zde výjimka, protože AVDP je dočasně umístěno na sektoru 512 nebo 256 až

Adresa	Délka [B]	Jméno položky	Datový typ	Popis
0	16	Descriptor Tag	tag	Popis deskriptoru
16	8	Main Volume Descriptor Sequence Extent	extent_ad	Sektor a délka *Main Volume Descriptor Sequence*
24	8	Reserve Volume Descriptor Sequence Extent	extent_ad	Sektor a délka *Reserve Volume Descriptor Sequence*
32	480	Rezerva		

Tab. 2.2: Anchor Volume Descriptor Pointer

do uzavření média, kdy je dopsáno na svá místa a opět alespoň ve dvou exemplářích, přičemž AVDP na adrese 512 se do tohoto počtu nezapočítává.

Jeho struktura je popsána v tabulce 2.2. Jak je vidět, obsahuje ukazatele na hlavní a záložní VDS, což zvyšuje robustnost celého souborového systému.

2.3.3 Skupina deskriptorů Volume Descriptor Sequence

Volume Descriptor Sequence (VDS) je skupina deskriptorů popisující veškeré vlastnosti souborového systému UDF. Jsou seřazeny postupně na jednotlivých sektorech počínaje adresou udanou z AVDP (viz. kapitola 2.3.2) a končí po počtu logických sektorů uloženém opět v AVDP. Jsou uloženy postupně po sektorech od výchozí adresy a končí *Terminating Descriptor*, což je prázdný deskriptor, který obsahuje pouze *Descriptor Tag*. Samotné pořadí deskriptorů není pevné a samotné deskriptory jsou identifikovány právě pomocí *Descriptor Tag*.

Jak je vidět v AVDP, i tato část je uložena duplicitně pro zvýšení robustnosti. Rezervní VDS je uložena ve stejném tvaru jako primární, tentokrát však od jiné adresy, opět uvedené v AVDP.

Samotné deskriptory jsou popsány ve standardu [2], kapitola 2.2. Pro kompletnost zde uvedu jejich výčet ze standardu verze 2.01.

- *Primary Volume Descriptor*
- *Logical Volume Descriptor*
- *Unallocated Space Descriptor*
- *Implementation Use Volume Descriptor*
- *Partition Descriptor*
- *Terminating Descriptor*

Z nich je ovšem vhodné některé vyzdvihnout a popsat hlouběji kvůli přímému vlivu na funkci `fsck`.

Adresa	Délka [B]	Jméno položky	Datový typ	Popis
0	16	Descriptor Tag	tag	Identifikátor deskriptoru
16	4	Volume Descriptor Sequence Number	Uint32	
20	64	Descriptor Character Set	charspec	
84	128	**Logical Volume Identifier**	dstring	**Jméno logického svazku**
212	4	**Logical Block Size**	Uint32	**Velikost logického bloku**
216	32	Domain Identifier	regid	
248	16	**Logical Volume Contents Use**	bytes	**Umístění File Set Descriptor**
264	4	Map Table Length (=MT_L)	Uint32	
268	4	Number of Partition Maps	Uint32	
272	32	Implementation Identifier	regid	ID vývojáře a operačního systému, který svazek vytvořil
304	128	Implementation Use	bytes	
432	8	**Integrity Sequence Extent**	extent_ad	**Umístění LVID**
440	MT_L	Partition Maps	bytes	

Tab. 2.3: Logical Volume Descriptor. Tučně zvýrazněné části jsou kritické pro činnost nástroje pro detekci a korekci chyb.

Logical Volume Descriptor

Logical Volume Descriptor (LVD) popisuje logický svazek jako celek. Pro použití ve `udfffsck` jsou důležité tyto:

- *Logical Block Size* - velikost logického bloku svazku. Měla by být shodná s velikostí sektoru. Pokud není, je to v rozporu se specifikací a kontrola není možná.
- *Integrity Sequence Extent* - zde je uloženo umístění *Logical Volume Integrity Descriptor* 2.3.4.
- *Logical Volume Identifier* - pojmenování logického svazku. Je to řetězec komprimovaného Unicode s maximální možnou délkou 128 B.
- *Logical Volume Contents Use* - zde je uloženo umístění *File Set Descriptor* 2.4.1.

Struktura deskriptoru je popsána v tabulce 2.3.

Partition Descriptor a Space Bitmap Descriptor

Partition Descriptor (PD) popisuje vlastnosti logického svazku. Důležitým údajem je položka *Partition Contents Use*, která popisuje jak se má zacházet s obsahem

Adresa	Délka [B]	Jméno položky	Datový typ	Popis
0	16	Descriptor Tag	tag	Identifikátor deskriptoru
16	4	Volume Descriptor Sequence Number	Uint32	
20	2	Partition Flags	Uint16	
22	2	Partition Number	Uint16	
24	32	**Partition Contents**	regid	**Definuje obsah logického svazku**
56	128	**Partition Contents Use**	bytes	**Obsahuje *Partition Header Descriptor***
184	4	**Access Type**	Uint32	**Určuje filozofii zápisu**
188	4	Partition Starting Location	Uint32	
192	4	Partition Length	Uint32	
196	32	Implementation Identifier	regid	
228	128	Implementation Use	bytes	
356	156	Rezerva		

Tab. 2.4: Partition Descriptor. Tučně zvýrazněné části jsou kritické pro činnost nástroje pro detekci a korekci chyb.

logického svazku. Obsah logického svazku je definován položkou *Partition Contents*, která může nabývat hodnot uvedených v tabulce 2.5. Další důležitý údaj je v položce *Access Type*, který určuje filozofii přístupu k zápisu na logický svazek (více v kapitole 2.2.)

Nejdůležitější z vyjmenovaných parametrů je *Partition Contents Use*. Ten v sobě podle [2], kapitola 2.3.3 nese *Partition Header Descriptor*, který obsahuje adresu *Unallocated Space Bitmap*, což je *Space Bitmap Descriptor* (SBD).

Ten obsahuje bitový vektor reprezentující logický svazek, kde každý bit odpovídá jednomu logickému bloku. Ten může nabývat dvou hodnot, volný (logická 1) nebo obsazený (logická 0). Spolu s tím je uložen počet bytů a bitů. Tyto dva údaje jsou kritické pro správné fungování bitového vektoru. Důvod je granularita vektoru, která je odvozena od nejmenší možné jednotky a tou je byte, který obsahuje 8 bitů. Tím pádem, pokud oddíl (a potažmo vektor) nemá velikost dělitelnou beze zbytku osmi, zůstanou některé bity v posledním bytu nevyužité. To je nutné indikovat právě počtem bitů. Z tohoto vyplývá, že počet bitů může být buď stejný jako počet bytů nebo nanejvýš o 7 větší.

Důležitost tohoto deskriptoru je při detekci zaalokovaného místa, které není nikde využito, tudíž indikace nedokončeného zápisu.

Struktura PD je zachycena v tabulce 2.4. Struktura SBD je zachycena v tabulce 2.6.

Obsah	Interpretace
+FDC01	Médium bylo vytvořeno podle standardu ECMA-107
+CD001	Médium bylo vytvořeno podle standardu ECMA-119
+CDW02	Médium bylo vytvořeno podle standardu ECMA-168
+NSR03	**Médium bylo vytvořeno podle standardu ECMA-167/UDF**

Tab. 2.5: Partition Contents reprezentace. Zvýrazněný identifikátor je jediný, který nástroj bude podporovat.

Adresa	Délka [B]	Jméno položky	Datový typ	Popis
0	16	Descriptor Tag	tag	Identifikátor deskriptoru
16	4	Number of Bits (=N_BT)	Uint32	Počet použitých bitů
20	4	Number of Bytes (=N_B)	Uint32	Velikost bitmapy
24	N_B	Bitmap	bytes	Samotná bitmapa

Tab. 2.6: Space Bitmap Descriptor

2.3.4 Skupina deskriptorů Logical Volume Integrity Sequence

Další důležitou částí je *Logical Volume Integrity Descriptor* (LVID) spolu s *Terminating Descriptor* (TD). Ty dohromady vytváří *Logical Volume Integrity Sequence*. Tato sekvence je uložena pouze jednou a její umístění je udržováno v LVD. Odpovídá na následující otázky:

- Je Logický svazek v konzistentním stavu?
- Kdy bylo naposledy cokoli na Logickém svazku modifikováno.
- Kolik bloků je volných na svazku?
- Jaká je celková velikost svazku v blocích?
- Byl obsah modifikován nějakou *jinou* implementací od posledního přístupu implementace která svazek vytvořila?

Jak je vidět z tohoto výčtu, tato část je důležitá hlavně pro čtení a zápis, ale i pro kontrolu dat má svůj význam, například právě kvůli informaci o konzistenci svazku.

Samotný formát struktury LVID je opět v [2], kapitola 2.2.6 a v tabulce 2.7.

Pokud bychom měli zvýraznit některé položky, byly by to tyto:

- *Recording date and time* - časová značka posledního záznamu, který byl proveden na médiu.
- *Integrity Type* - stav svazku z hlediska probíhajících operací. Může být *Open* (médium je používáno, možný probíhající zápis) nebo *Close* (všechny trasakce byly dokončeny a médium je v konzistentním stavu.)
- *Free Space Table* - tabulka popisující volné místo na jednotlivých logických svazcích v logických blocích.
- *Size Table* - tabulka popisující velikost jednotlivých logických svazků v logic-

Adresa	Délka [B]	Jméno položky	Datový typ	Popis
0	16	Descriptor Tag	tag	Identifikátor deskriptoru
16	12	**Recording Date and Time**	timestamp	**Časová značka záznamu**
28	4	**Integrity type**	Uint32	**Otevřené nebo uzavřené médium**
32	8	Next Integrity Extent	extent_ad	Umístění dalšího LVID
40	32	Logical Volume Contents Use	bytes	
72	4	Number of Partitions (=N_P)	Uint32	Počet oddílů
76	4	Length of Implementation Use (=L_IU)	Unit32	
80	N_P×4	**Free Space Table**	Unit32	**Tabulka volného místa**
N_P×4 + 80	N_P×4	**Size Table**	Uint32	**Tabulka velikostí**
N_P×4 + 80	L_IU	**Implementation Use**	bytes	**Počet adresářů a souborů a povolených verzí UDF**

Tab. 2.7: Logical Volume Integrity Descriptor. Tučně zvýrazněné části jsou kritické pro činnost nástroje pro detekci a korekci chyb.

kých blocích.

- *Implementation Use* - oblast, nepopsaná ve standardu ECMA-167, byla standardem UDF použita k uložení počtu adresářů a souborů uložených na médiu. Dále obsahuje povolené verze UDF pro práci s médiem.

2.4 Struktura souborového stromu

Souborová struktura UDF je postavena na třech deskriptorech, které se podílí na popisu a uložení dat. Jedná se o tyto deskriptory:

- *File Set Descriptor* (FSD)
- *File Entry* (FE)
- *File Identifier Descriptor* (FID)

Ty jsou podporovány celou řadou dalších deskriptorů. Z nich je potřeba některé popsat:

- *Information Control Block* (ICB) – deskriptor popisující umístění samotných dat. Jedná se o nadřazené označení pro *File Entry* a *Extended File Entry*. Tuto funkci doplňuje ICB tag, který popisuje metadata ICB.
- *Allocation Descriptor* (AD) – deskriptory popisující umístění v rámci logického svazku. Existují tři varianty: *Short* (ukazauje v rámci logického svazku), *Long*

(ukazuje na libovolný logický svazek) a *Extended* (ukazuje na libovolný logický svazek a nese navíc informaci o skutečné délce obsahu, ostatní nesou pouze informaci o počtu použitých bloků.)

S touto sadou deskriptorů je již možné popsat strukturu souborového stromu.

2.4.1 File Set Descriptor

File Set Descriptor (FSD) je výchozím bodem pro čtení dat. Jeho pozice je opět uložena v LVD (více v kapitole 2.3.3). Jeho struktura je popsána v [2], kapitola 2.3.2. Tento deskriptor obsahuje hlavně umístění kořenového adresáře a název svazku, na kterém je uložený. Další údaje uložené v tomto deskriptoru jsou umístění souborů obsahující copyright nebo abstrakt. Ty stojí mimo adresářovou strukturu a jsou využívány například na DVD s filmy.

Kořenový adresář je jediný, který není uveden pomocí *File Identifier Descriptor*, ale přímo pomocí ICB mířícího na *File Entry*.

2.4.2 Deskriptor File Entry

File Entry (FE) je deskriptor, který, jak naznačuje název, popisuje jednotlivé záznamy v oblasti dat. Ke každému jednomu souboru nebo adresáři patří jeden FE. Jeho umístění je určeno jeho *rodičovským adresářem*, respektive ji určuje *File Identifier Descriptor*, který je v něm uložen.

Z tohoto pravidla existuje jedna výjimka a to v případě kořenového adresáře. Ten nemá nadřazený adresář (sám je nejvýše umístěný adresář) a jeho pozice je určena přímo pomocí FSD. To s sebou nese druhou výjimku a tou je *Unique ID*. Ta nabývá hodnot 1 až 2^{64}, přičemž 0 je vyhrazena pro použití před nastavením *Unique ID* a pro kořenový adresář, který jediný musí mít *Unique ID* rovno 0.

Jeho struktura je popsána v [2], kapitola 2.3.6, a z ní pouze vyberu některé důležité části.

První důležitou informací je počet zapsaných bloků *Logical Blocks Recorded*. Tento údaj pomáhá vytvořit mapu obsazených sektorů pro srovnání s *bitmap* z SBD 2.3.3 a také skutečný počet obsazených sektorů pro srovnání s *Free Space Table* v LVID 2.3.4.

Klíčový údaj pro další práci s FE je rozlišení typu FE. K tomu slouží *ICB tag*, který obsahuje položku *File Type*. Ta může nabývat celé řady hodnot, zajímavé pro nás jsou dvě a to *Directory* a *Regular*. Právě údaj z této části pomáhá rozlišit adresáře od souborů. Další hodnoty jsou speciální druhy záznamů, jako jsou symbolické odkazy, stream adresáře a podobné.

Další důležitou částí jsou *Allocation Descriptors*. Velikost této části je určena pomocí *Length of Allocation Descriptors* a obsahuje dva druhy dat, podle toho, jestli se FE popisuje adresář nebo soubor. V případě souboru je v tomto místě uložena pozice dat souboru ve formě *Long Allocation Descriptor* (`long_ad`) nebo *Short Allocation Descriptor* (`short_ad`). Od adresy uvedené v těchto deskriptorech jsou sekvenčně za sebou ukládána data. V případě adresáře jsou v *Allocation Descriptors* uloženy *File Identifier Descriptor* (kapitola 2.4.3), přičemž každý jeden deskriptor nese informaci o jednom *potomkovi* adresáře. Vždy je přítomen nejméně jeden FID a tím je *rodičovský adresář* aktuálního FE.

Zde je opět výjimka týkající se kořenového adresáře. Tou je reference na *rodičovský adresář*, která nemůže vést na FSD, tak je zavedena autoreferenčně na FE kořenového adresáře.

Posledním faktem, který je nutno zdůraznit u deskriptoru FE, je existence novější verze *File Entry* a tou je *Extended File Entry* (EFE). Ta přišla s verzí UDF 2.00 a je doporučené je používat místo FE. Ovšem pro zachování zpětné kompatibility je možné nechat FE na médiu a nadále je používat a nové soubory ukládat pomocí EFE. To s sebou nese nutnost rozlišovat mezi FE a EFE v rámci média, protože EFE pouze nerozšiřuje FE, ale mění i jeho strukturu, takže původní položky jsou sice shodně pojmenované a nesou stejnou informaci, ale mají jiné umístění.

2.4.3 File Identifier Descriptor

File Identifier Descriptor (FID) je struktura, která udržuje informace o souboru na úrovni mateřského adresáře. Je popsaný v [2] v kapitole 2.3.4 a v tabulce 2.8.

Účelem FID je zrychlení přístupu k metadatům souborů na sekvenčních médiích, kde přesun na jiné místo trvá delší dobu než u zařízení s náhodným přístupem (*Random Access*). Takto jsou základní údaje o souboru (adresáři) shluknuty dohromady v rámci rodičovského adresáře a není potřeba je hledat na jiných místech média.

Z deskriptoru FID stojí za zvýraznění *File Identifier*, který udržuje jméno záznamu. Dále se jedná o *File Characteristics*, které popisují vlastnsti záznamu z hlediska práce s ním. Kompletní výčet možností je v tabulce 2.4.3. Položka, kterou je také vhodné zmínit, je *Unique ID*. To by mělo být shodné s hodnotou, která je uložená v FE, na které ukazuje. Nejdůležitějším údajem je samozřejmě pozice samotného FE (nebo EFE). Ta je uložena v položce *ICB*.

2.5 Zhodnocení kapitoly

V této kapitole byly nastíněny základní principy UDF, jeho strukutra a vlastnosti. Protože není cílem popsat existující dokumenty, není výčet kompletní a jedná se

Adresa	Délka [B]	Jméno položky	Datový typ	Popis
0	16	Descriptor Tag	tag	Identifikátor deskriptoru
16	2	File Version Number	Uint16	Mělo by být nastaven na 1
18	1	**File Characteristics**	Uint8	**Charakteristiky souboru (adresář, smazaný atp.)**
19	1	Length of File Identifier (=L_FI)	Uint8	Délka jména
20	16	**ICB**	long_ad	**Umístění deskriptoru FE**
36	2	Length of Implementation Use (=L_IU)	Uint16	
38	L_IU	Implementation Use	bytes	Identifikátor poslední implementace, která se souborem pracovala
L_IU+38	L_FI	**File Identifier**	d-characters	**Jméno v Unicode**
L_FI+L_IU+38	*	Padding	bytes	Zarovnání délky na N mod 4

Tab. 2.8: File Identifier Descriptor. Tučně zvýrazněné části jsou kritické pro činnost nástroje pro detekci a korekci chyb.

Bit	Význam
0	**Existence** – Pokud je nastavený, existence souboru by neměla být známá uživateli.
1	**Directory** – Pokud je nastavený, záznam reprezentuje adresář.
2	**Deleted** – Pokud je nastavený, záznam je smazaný.
3	**Parent** – Pokud je nastavený, záznam je rodičovský vůči tomuto.
4	**Metadata** – Pokud je nastavený a je ve *Stream directory*, záznam obsahuje metadata.
5	Rezervováno, mělo by být rovno 0.

Tab. 2.9: Význam jednotlivých bitů položky File Characteristics deskriptoru FID 2.8

pouze o přehledovou část, ve které jsou zvýrazněny části důležité pro kontrolu a opravu konzistence souborového systému.

Výchozími dokumenty pro tuto kapitolu jsou standardy OSTA UDF do verze 2.01 [2] a jemu nadřazený standard ECMA-167, revize 3 [4] z důvodů uvedených v podkapitole 3.1. Existuje sice novější verze UDF 2.60, ale ta není v práci použita. Veškeré deskriptory byly popsány v duchu těchto dokumentů ve výše zmíněných verzích.

3 NÁSTROJE PRO KONTROLU KONZISTENCE UDF

Pro OS Linux je znám pouze nástroj `udfct_1.5r4`. Tento nástroj byl vyvinut firmou Philips a jedná se o nástroj, který je schopný zkontrolovat integritu souborového systému UDF. Chyby v implementaci UDF a případné chyby v datech vypisuje do terminálu, ale není schopný je opravovat. Zásadním problémem tohoto nástroje je, že existuje pouze pod restriktivní licencí společnosti Philips a není tudíž možné jej využít jako výchozí bod dalšího vývoje i přes dostupnost zdrojových souborů. V současnosti je balíček se zdrojovými kódy dostupný již pouze díky službě Wayback Machine [7], zástupci společnosti Philips jej již nenabízí. Mou žádost o poskytnutí práva na přepoužití jejich kódu ignorovali.

V BSD je `fsck` v podobném stavu jako v Linuxu ovšem komunita okolo BSD na problému pomalu pracuje. Našel jsem blog [11] od Scotta Longa z projektu FreeBSD, kde má shrnutí své práce na UDF včetně jeho patchů do FreeBSD implementace UDF. Mezi jeho body k doplnění je i nástroj `fsck` a právě proto jsem ho oslovil s prosbou o informace. Právě Scott Long mne odkázal na projekt NetBSD, protože on sám nikdy na `fsck` nezačal pracovat, ale doslechl se, že v projektu NetBSD něco vzniká. Po hledání jsem nalezl mailing list z roku 2008, kde Reinoud Zandijk popisuje svůj postup práce na UDF implementaci. Na konci jeho zprávy je poznámka o `fsck` s informací, že bude "brzy". Žádná implementace se ovšem na svět nedostala, takže jsem mu napsal s prosbou o informace o stavu jeho implementace `fsck`. Jeho odpověď byla vyčerpávající a potvrzovala mé podezření o stavu open source implementace. On sám má rozpracovanou implementaci `fsck`, ale zveřejňovat ji zatím nebude, protože není dokončena, jeho slovy:

> „I've created two UDF implementations: UDFclient [12] and the NetBSD UDF implementation. UDFclient was a kind of study into UDF and far too elaborate to be useful for my FSCK implementation, it'll need to be pruned first.“

Nikdo další žádnou jinou open source implementaci nemá vyjma projektu Open Solaris, což bylo panem Reinoudem Zandijkem okomentováno takto:

> „As for `fsck_udf`, there is only one opensource version and thats the OpenSolaris one. Before you get too excited, its fairly limited and will only check older media and even then only deals with recovering free space and get the directory tree in-order. Important but limited.“

Implementace v projektu Open Solaris je tedy v současnosti jedinou dostupnou open source implementací `fsck`. Jejich kód je dostupný na serveru GitHub [13]. Bohužel

ani jejich implementace není kompletní a dokáže obnovit pouze volné místo a strom souborového systému ale bez dat. Dalším omezením je maximální verze UDF, která je omezena na verzi 2.01 ovladačem UDF. Nástroj jako takový exaktní omezení verze nemá. Ovšem i toto je dobrý výchozí bod pro další práci.

Co se týká OSX od společnosti Apple, ti mají nástroje pro kontrolu UDF implementovanou v nástroji `fsck_udf` pro všechny existující revize, ovšem zdrojové kódy nejsou veřejně dostupné, takže je možné jejich nástroje použít pouze jako referenční pro srovnání funkčnosti. Druhým nedostatkem je absence jakýchkoli korekcí.

Microsoft Windows má také implementovaný `CHKDSK` pro UDF, ale jeho zdrojový kód je bohužel uzavřený.

3.1 Stav projektu udftools

Projekt udftools byl založen roce 1999 na projektovém serveru SourceForge [8] Bennem Fennemou. Byl implementován standard UDF 1.5 a byly vytvořeny nástroje pro paketový zápis na CD, vytvoření souborového systému UDF a pro přístup k němu. Nástroj pro kontrolu konzistence zůstal jako prázdný projekt, byly ovšem vytvořeny nástroje pro vytvoření souborového systému UDF, nástroje pro jeho zápis včetně paketového zápisu (nástroj pro zapisování na CD-R přímo z OS bez nutnosti volat další programy.)

V roce 2007 byly integrovány poslední záplaty a poté projekt zůstal ležet ladem. V roce 2014 byl projekt přemigrován na GitHub [9] Palim Rohárem a znovu byl započat vývoj, převážně opravy starých chyb. Byla implementována verze UDF 2.01. Nástroj `fsck.udf` nebyl v tomto projektu nikdy vytvořen a tento stav přetrvává do současnosti.

V tuto chvíli se o projekt udftools stará Pali Rohár, který je správce projektu. Vzhledem k nepříliš vysoké popularitě UDF není projekt udftools aktivní. Sice je stále udržován panem Rohárem, ale aktivní vývoj v tuto chvíli pravděpodobně neprobíhá, nebo alespoň ne veřejně.

Protože bude moje práce navazovat na tento projekt, bude v tuto chvíli maximální možná verze omezena na UDF 2.01, stejně jako zbytek balíčku. Důvodem je sdílená knihovna se zbytkem balíčku obsahující hlavičky deskriptorů, kde by zásah způsobil změny v ostatních nástrojích a také nedostatek testovacích médií ve vyšších verzích.

4 DEFINICE MOŽNÝCH CHYB SOUBOROVÉHO SYSTÉMU

Chyby na souborovém systému mohou vzniknout ze třech příčin.

1. Chybou ovladače souborového systému.
2. Nekorektním odpojením souborového systému (například odpojení flash disku před ukončením všech transakcí).
3. Fyzickým poškozením média (například stářím poškozené bloky flash paměti nebo poškrábané optické médium).

První druh chyb se děje zřídka. Důvodem je skutečnost, že programy a kernelové moduly starající se o přístup k a práci se souborovými systémy bývají dobře odladěné a otestované. Koneckonců, právě data jsou to, co má v počítačích hodnotu.

Chyby vzniklé nekorektním odpojením souborového systému se vyskytují velice často. Odebrání disku ve spěchu bez korektního odpojení může způsobit poškození souborového systému skrz přerušení probíhající zápisové operace. Systém poté zůstane v nekonzistentním stavu, protože se v něm nachází částečně zapsaný soubor. Případně může dojít k poškození metadat souboru. Do této kateogie spadají i chyby vzniklé havárií operačního systému. Toto riziko se ještě zvýšilo s USB flash disky, kde je velice výrazný vliv vyrovnávacích pamětí jak na straně řadiče, tak na straně disku a i když je disk odebrán ze systému, může ještě probíhat zápis právě z vyrovnávací paměti do flash paměti.

Třetí kategorií chyb jsou veškeré poruchy fyzického média. U optických disků a disket se první vybaví škrábance a rýhy. Tyto chyby poté bývají seskupeny do clusterů poškozených dat. U magnetických disků může dojít k poškození kolizí čtecích hlav s plotnou nebo k poškozením opotřebením.

Existuje určitá prevence některých těchto poruch. Vyšší řady notebooků mají integrovaný akcelerometr a v případě většího zrychlení dojde k nouzovému zaparkování čtecích hlav harddisku. Ovšem ani tato ochrana není absolutní a spíše zabraňuje samotné činnosti disku při pohybu (například při chůzi). Chybám z opotřebení lze předcházet pomocí integrovaného systému S.M.A.R.T., který se stará o sběr telemetrických dat o disku a na jejich základě lze předvídat jeho poruchu (ovšem ani zde není stoprocentní záruka.)

Opravitelnost a analýza chyb je vždy otázkou míry a typu poškození. Od určité míry poškození je oprava buď nemožná, nebo ne úplná. Tuto skutečnost je třeba mít na paměti po celou dobu návrhu opravných algoritmů a je třeba si předem určit, jaký přístup zvolit, zda se spokojíme s částečnou opravou, nebo je oprava vyžadována pouze úplná.

V dále popsané implementaci byl zvolen kompromis, který se spíše přiklání k úplné opravě nebo žádné opravě. Z tohoto jsou vyjmuty pouze deskriptory VDS, které mohou být opraveny jen částečně (t.j. lze pokračovat i s některými deskriptory neopravitelnými.)

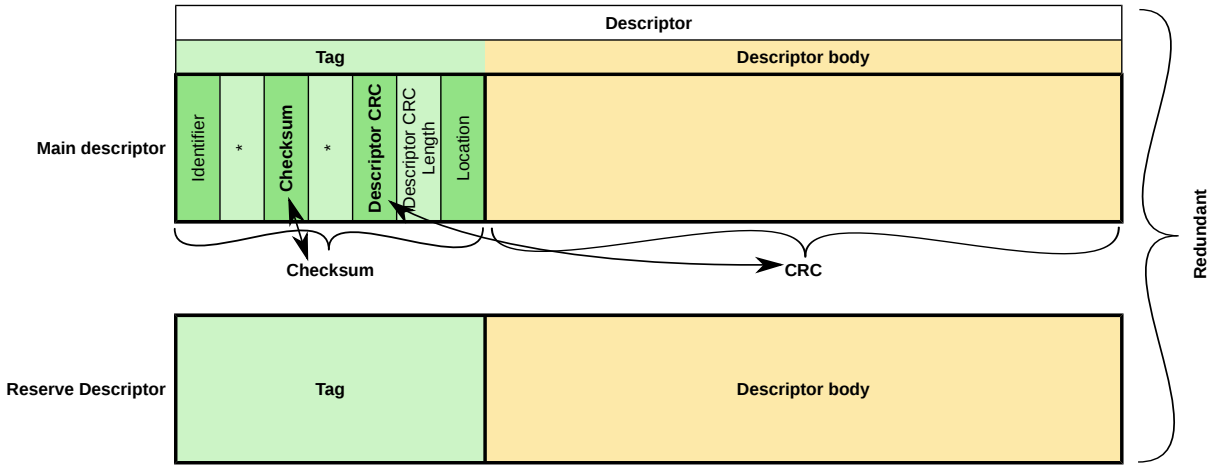
4.1 Kontrola správnosti dat

Způsobů, jak detekovat chyby v případě jejich výskytu, je celá řada a liší se požadavky na prostředky a čas. Za naivní přístup lze považovat například detekci porovnáním s referencí. Předpokládejme, že máme stejnou informaci uloženou dvakrát na různých místech (médiích) a v případě poruchy dat je možné data obnovit z referenčního média. Toto řešení je ale evidentně náročné na úložiště kvůli stoprocentní duplicitě dat. Výměnou za to poskytuje informaci nejen o přítomnosti chyby, ale i o její poloze a původních datech. Předpokladem ovšem je znalost, které médium obsahuje data uložená správně.

Dalším přístupem je vytváření kontrolních součtů. Tento přístup nám dává oproti předchozímu pouze informaci o přítomnosti chyby bez informace, kde chyba je a jak ji opravit. V extrémním případě se můžou chyby dokonce navzájem vyrušit, takže data nebudou konzistentní, ale kontrolní součet bude stejný. I přes tyto nedostatky je kontrolní součet oblíbená metoda detekce chyb kvůli své jednoduchosti a nenáročnosti na paměť.

Poněkud komplexnější variantou kontrolního součtu je CRC (*Cyclic Redundancy Check*). Tento mechanismus, který původně vznikl pro kabelové přenosy, našel uplatnění všude, kde je potřeba větší šance na detekci chyby než při použití kontrolního součtu. CRC je na výpočet složitější, jeho výstupem je několikabytové číslo a důvodem je větší citlivost na chyby a nižší šance na stejný výsledek při chybě v datech.

Pokud se tyto metody zkonkretizují pro použití na UDF, zjistíme, že jsou použity všechny, ovšem nikoli jednotlivě, ale v kombinaci. To zvyšuje robustnost celého řešení. Tato logika je zachycena na obrázku 4.1. Na obrázku jsou dva deskriptory, hlavní a záložní (případ u VDS). Každý z nich (naznačeno to je pro přehlednost pouze u hlavního) je zabezpečen kombinací CRC a kontrolního součtu následujícím způsobem: Tag, který identifikuje deskriptor, je zabezpečen kontrolním součtem a nese si referenční hodnotu sám v sobě (aby nedošlo k rekurzivní chybě při výpočtu, je místo s referenční hodnotou kontrolního součtu pro jeho výpočet vynecháno). Zároveň zabezpečuje referenční hodnotu CRC pro celé tělo deskriptoru. Pokud je jak kontrolní součet, tak CRC v pořádku, máme určitou jistotu, že deskriptor není poškozen. Tudíž lze pro obnovu vybrat nepoškozený deskriptor jako vzor, pokud jsou k dispozici dva exempláře.



Obr. 4.1: Schéma kombinace detekčních mechanismů UDF

Nutno podotknout, že toto pouze zajišťuje správnost z hlediska poškození média, nikoli algoritmickou chybu ovladače. Pokud ovladač změní obsah deskriptoru a správně přepočítá CRC a checksum, tento způsob detekce tuto chybu nemůže odhalit.

4.1.1 Zabezpečení pomocí kontrolního součtu

Zabezpečení pomocí kontrolního součtu (*checksum*) patří k nejzákladnějším kontrolním mechanismům.

Mějme číslenou sekvenci, nad kterou chceme vypočítat kontrolní součet. Postupně přičítejme k výchozí hodnotě (typicky 0) hodnoty z číselné sekvence. Výsledné číslo je kontrolní součet, neboli součet všech hodnot původní sekvence.

V implementaci ve výpočetní technice je toto obvykle realizováno nad sekvencí z nejmenšího datového typu, t.j. nad bytem, který může nabývat hodnot $< 0; 2^8 - 1 >$. Datový typ kontrolního součtu bývá také o velikosti 1 B, ale může být i větší. Pokud hodnota součtu překoná maximální velikost použitého datového typu, výsledek je oříznut na N-bitů a jsou zachovány jeho méně významné bity.

Příklad výpočtu kontrolního součtu pro 5 B dlouhou sekvenci $S = \{10, 25, 130, 221, 85\}$ a 1 B checksum C. V tabulce 4.1.1 je příklad výpočtu nad sekvencí S. Kontrolní součet je tedy v tomto případě $C = 89$.

4.1.2 Zabezpečení pomocí Cyclic Redundancy Check

Cyclic Redundancy Check (CRC) je složitější varianta kontrolního součtu. Matematické odůvodnění jeho funkce je v [36] (přednášky č.6, 7, 8 a 11.)

Iterace i	Byte B	Kontrolní součet: $C_i = (C_{i-1} + B) \bmod 256$	Popis
0	–	0	Výchozí stav
1	10	10	
2	25	35	
3	130	165	
4	221	4	Zde došlo k přetečení a oříznutí na 8 nižších bitů
5	85	89	Konec výpočtu

Tab. 4.1: Příklad výpočtu kontrolního součtu

Důvody pro nasazení CRC oproti kontrolnímu součtu jsou hlavně odolnost vůči chybám vzniklým prohozením bytů, které kontrolní součet z evidentních důvodů nezachytí, zatímco CRC ano. Dalším důvodem je různá výsledná hodnota pro různé délky vstupních dat, přestože jsou všechny vstupní hodnoty nulové.

Ústředním bodem CRC je generační polynom GP. Ten určuje, na jaký druh chyb bude CRC citlivé, zdůvodnění opět v [36]. Základní operací, která je při výpočtu CRC použita, je logická operace XOR, která postupně maskuje vstupní sekvenci s cílem ji vynulovat. Z tohoto důvodu je GP vždy zarovnaný nejvíce významným bitem s nejvíce významným bitem dat.

Samotný výpočet je naznačen v tabulce 4.1.2. Je zde aplikován výpočet 4-bitového CRC s generačním polynomem $GP = 1 \cdot x^3 + 0 \cdot x^2 + 1 \cdot x^1 + 0 \cdot x^0$, neboli dle koeficientů *GP=1010*. Pro ukázku jsou použita vstupní data `0100 1011 0001 1111 1100`. Ta jsou rozšířena o délku požadovaného CRC, tedy o čtyři bity s hodnotou 0 (viz. iterace 0 v tabulce.) Podtržená část naznačuje aktuální umístění GP, t.j. které bity jsou aktuálně eliminovány.

Na konci výpočtu zbude 4-bitové číslo, které je hledaná hodnota CRC. Detekce, zda došlo k chybě, probíhá stejným způsobem. K datům je připojena část o délce CRC a je vypočítáno znovu. Pokud se výsledky shodují, data jsou v pořádku.

4.2 Použité kontrolní mechanismy v UDF

V UDF se využívá několika různých mechanismů, které jsou v případě nutnosti kombinovány pro zvýšení robustnosti. Z obecných mechanismů se jedná o tyto:

- redundace dat na médiu,
- kontrola deklarované a skutečné polohy deskriptoru,
- kontrolní součet (checksum),
- robustní kontrolní součet (CRC).

Iterace i	Vstupní slovo	CRC: $C_i = C_{i-1} \bigoplus GP(1010)$	Popis
0	1001	0100 1011 0001 1111 1100 0000	Výchozí stav
1	1101	0001 1011 0001 1111 1100 0000	První XOR s GP
2	1111	0000 1111 0001 1111 1100 0000	
3	1010	0000 0101 0001 1111 1100 0000	
4	1111	0000 0000 0001 1111 1100 0000	
5	1011	0000 0000 0000 1011 1100 0000	
6	1110	0000 0000 0000 0001 1100 0000	
7	1000	0000 0000 0000 0000 1000 0000	
8	1000	0000 0000 0000 0000 0010 0000	
9	1000	0000 0000 0000 0000 0000 1000	Výsledné CRC

Tab. 4.2: Příklad výpočtu *Cyclic Redundancy Check* s $GP = 1 \cdot x^3 + 0 \cdot x^2 + 1 \cdot x^1 + 0 \cdot x^0$

V tomto výčtu jsou zachyceny všechny obecné kontrolní mechanismy na úrovni deskriptorů UDF. Kontrolní součty jsou popsány v kapitole 4.1.1, CRC je popsáno v kapitole 4.1.2.

Kontrolní součet je použit pro kontrolu *DescriptorTag* a CRC pro kontrolu samotného těla deskriptoru, přičemž jeho referenční výsledek je uložený v již zkotrolovaném tagu. Tento princip je zachycen v kapitole 4.1.

Redundance deskriptorů je mechanismus, kterým standard ECMA-167 ošetřil nejdůležitější deskriptory souborového systému a to AVDP (více v kapitole 2.3.2) a skupinu VDS (více v kapitole 2.3.3). V případě AVDP byla redundance ještě podtržena fyzickou vzdáleností, kdy je první AVDP uloženo na sektoru 256 a druhý na konci média.

Vedle těchto obecných mechanismů jsou implementovány některé další, konkrétní pro UDF, byť stále vycházející ze standardu ECMA-167.

- Použití *Unique ID* pro zabezpečení párovatelnosti metadat souborů k patřičným adresářům. Více v kapitole 5.13.
- Udržování časových značek indikující poslední práci se souborovým systémem. Více v kapitole 5.11.
- Značka v položce *Integrity Type* indikující probíhající zápisovou operaci na médium. Více v kapitole 5.11.
- Pořadí operací pří zápisu na médium.

Pořadí úkonů zápisové operce na médium je toto:

1. Nastavení probíhající zápisové operace v LVID *Integrity Type* na *Open*.
2. Kontrola a změna objemu volného místa v *Free Space Table* a v SBD.
3. Zvýšení počtu souborů / adresářů v LVID.
4. Spuštění zápisu dat na médium.

5. Po dokončení zápisu se vytvoří *File Entry* s patřičným *Unique ID* a časovou značkou.
6. Vytvoření nového *File Identifier Descriptor* v rodičovském adresáři a svázání s FE.
7. Ukončení zápisové operace nastavením *Integrity Type* na *Close*.

Toto pořadí úkonů zajišťuje, že při přerušení zápisu dat zůstane systém v nekonzistentním stavu a bude možné detekovat, že zápis byl nekorektně přerušen.

5 REALIZACE NÁSTROJE PRO DETEKCI CHYB

Nástroj `udffsck` vznikal ve dvou fázích. V první fázi byl navržen a implementován nástroj určený pouze k detekci chyb na UDF. Ten byl ve druhé fázi rozšířen o opravy nalezených chyb. Postup návrhu a následné implementace je popsán v následujcích podkapitolách.

5.1 Návrh nástroje pro detekci a korekci chyb

Návrh nástroje byl rozdělen do dvou částí a to na návrh detekce chyb a návrh jejich korekce.

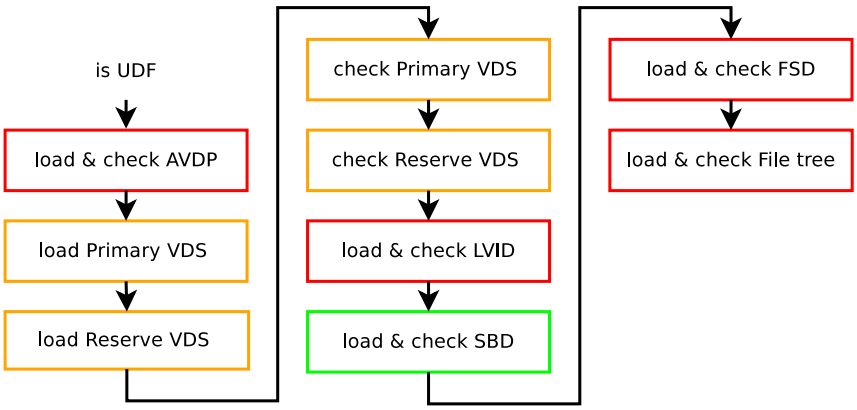
Vzhledem k faktu, že UDF existuje již řadu let a během té doby se postupně vyvíjelo, existuje celá řada potíží, které sebou UDF nese ve svých implementacích na různých platformách. Obvyklou příčinou je nedoimplementovaná nějaká oprava specifikace, která vyšla až po vydání daného nástroje. Otázkou je, jak nakládat s médiem, které trpí těmito problémy. Varianty jsou dvě: opravit podle poslední platné verze nebo nechat tyto problémy být, vzhledem k faktu, že se nejedná o provozní chyby, ale o chyby vůči specifikaci.

Byl zvolen benevolentnější přístup a to chyby vůči specifikaci, které vznikly až na základě oprav specifikace, neopravovat. Důvodem je častá nemožnost je opravit úplně. Typickým příkladem takovéto chyby je vyhrazená velikost deskriptoru LVID, který podle specifikace 2.01 má mít velikost 2048 B ale podle pozdějších dokumentů je doporučená velikost 8192 B. Problém je, že ne každé médium má místo pro umístění většího deskriptoru a poté není oprava možná. Proto byl tento druh chyb prozatím vynechán.

Chyby, které jsou detekovány vždy, jsou chyby v kontrolních mechanismech deskriptorů (jmenovitě kontrolní součet, CRC a umístění deskriptoru). Další druh chyb vzniká během nesprávného odebrání média. Těmi jsou chyby založené na mechanismech v kapitole 4.2, což jsou jmenovitě chyby typu neuzavřené médium, nenastavené nebo neshodující se *Unique ID*, špatná časová značka v deskriptoru souboru nebo v LVID. Tyto chyby jsou taktéž detekovány a považovány za chyby. Poslední druh chyb, které jsou detekovány, jsou neshodující se údaje o obsazeném a volném místě na médiu jednak z hlediska jeho velikosti a jednak z hlediska jeho umístění.

Na základě těchto požadavků vznikla struktura programu pro detekci těchto chyb. Její zjednodušená strukutura je zachycena na obrázku 5.1.

Zjednodušená z toho důvodu, že v ní není zachycena počáteční detekce velikosti sektoru a průchod stromem souborů je zjednodušen na jedinou operaci. Obecně



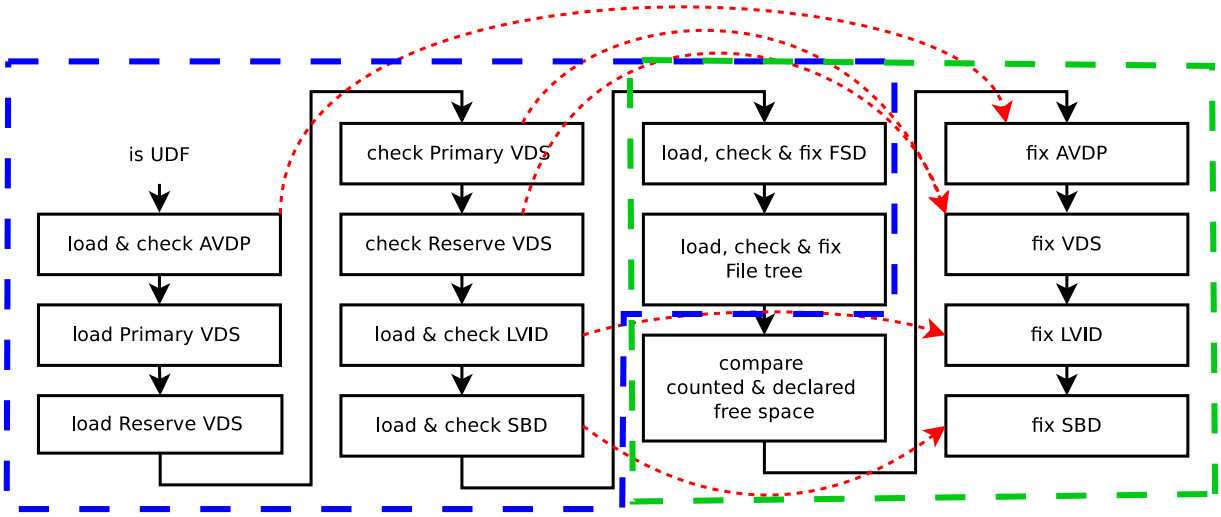
Obr. 5.1: Zjednodušený algoritmus detekce

se pod *Load* rozumí načtení deskriptoru z média do operační paměti a pod *Check* kontrola všech výše zmíněných chyb, pokud jsou v daném kroku aplikovatelné.

V algoritmu jsou kroky barevně rozlišeny do třech kategorií podle důležitosti, respektive podle jejich vlivu na další činnost pokud jejich načtení nebo kontrola selže. Červené kroky jsou kritické pro činnost a jejich selhání znamená selhání programu. Oranžové kroky jsou závislé na míře poškození. Pokud v PVD bude alespoň od každého druhu jeden deskriptor v pořádku, je možné pokračovat. Zelené kroky je možné vynechat, ale může to být na úkor některých detekcí (v případě SBD by byla vynechána kontrola rozložení obsazeného místa).

Návrh algoritmu pro korekci spočíval v navázání na existující detekční algoritmus. To spočívalo ve vytvoření mechanismu pro předání nalezených chyb druhé části algoritmu a způsobu jejich opravy. Tato část byla myšlenkově rozdělena opět na tři části a to na opravu deskriptorů jako takových, opravu volného místa a opravu stromu souborů. Struktura algoritmu korekce je zachycena na obrázku 5.2 V tomto obrázku, který navazuje na předchozí je cílem naznačit posloupnost jednotlivých operací a ukázat toky chybových stavů z detekcí ke korekcím (červené tečkované šipky). Modrý rámeček je část detekční, která je popsaná v předchozí části a zelený rámeček je část korekční. Jak je vidět, protínají se u práce s deskriptorem FSD a se souborovým stromem. Důvodem je předejití zbytečnému dvojímu procházení souborovým stromem. Proto jsou chyby v něm opravovány jak jsou nacházeny. Zbytek oprav již je ve svých oddělených krocích.

Důležitým detailem, který není v obrázku pro jednoduchost naznačen, je skutečnost, že každá korekce (t.j. zápis na médium) musí být autorizována uživatelem nebo apriori schválena při spuštění programu. Tudíž libovolná oprava může být přeskočena.



Obr. 5.2: Zjednodušený algoritmus detekce a korekce

5.2 Implementace nástroje pro detekci a korekci chyb

Implementace nástroje `udffsck` bude provedena v rámci balíčku `udftools`. Bude tak navazovat na existující sdílenou knihovnu, která obsahuje sdílené hlavičky s definicemi deskriptorů podle standardu ECMA-167.

Struktura nástroje bude rozdělena do třech částí. V první části bude probíhat načtení média a parsování vstupních parametrů. Druhá část bude detekovat přítomnost UDF, načítat deskriptory souborového systému a strom souborů a bude detekovat chyby v nich. Třetí část bude opravovat nalezené chyby na základě výsledků z předchozí části. Návrh jednotlivých částí je popsán v předchozí kapitole 5.1.

Pokud rozebereme implementaci na menší části, dostaneme se již na úroveň jednotlivých kroků implementace. Ta bude postupovat podle algoritmu popsaném v předchozí kapitole, viz obrázek 5.2. Bude vytvořena struktura pro přenos chybové informace mezi detekčními a korekčními algoritmy, stejně tak strkuktura nesoucí pracovní informace o médiu.

Nástroj bude vytvořen v jazyce C podle standardu C99 a bude využívat funkce poskytnuté prostředním GNU/Linux. Bude přeložitelný pomocí GCC pro ostré nasazení a pomocí LLVM CLANG pro vývoj. Důvodem je integrovaná funkce překladače LLVM CLANG *Address Sanitizer*, který poskytuje užitečné informace o správě paměti, respektive při jejím porušení. Díky tomu je snažší odhalit neodalokovanou paměť, přetečení mezi paměťovými oblastmi a v případě havárie programu vypíše užitečné informace kde v paměti k chybě došlo včetně zásobníku volání funkcí které k tomu vedlo. Dále potřeba použít debugger, zde bude použit GDB. K vytvoření testů

bude použita knihovna CMocka 1.1.0, ve které vzniknou v první řadě provozní testy celého programu, případně může být použita k tvorbě Unit Testů. Dokumentace k programu bude vytvořena pomocí nástroje Doxygen, který je určený k automatickému generování přehledné dokumentace z komentářů ve zdrojových kódech. Jako editor bude použitý Vim.

Vzhledem k závislostem zbytku balíčku (hlavně sdílené knihovny) je k překladu potřeba použít GNU Autotools spolu s knihovnou LibTool a Readline, která závisí na knihovně Ncurses. Tímto by měly být splněny požadavky pro implementaci.

Nástroj bude nutné testovat během vývoje vůči různým vzorkům souborových systémů UDF. K jejich vytvoření bude použit přednostně nástroj `mkudffs` z balíčku `udftools`, v menšině případů obrazy disků vytvořené na platformě Microsoft Windows a Apple macOS. Tyto obrazy disků budou následně poškozovány ať už cíleným přepisováním částí obrazů s cílem poškodit konkrétní části nebo budou nahrány nástrojem `dd` na flash disk a budou poškozovány odpojením disku během operace s médiem.

Závěrem implementační fáze bude začlenění výsledného `udffsck` zpět do balíčku `udftools`, ze kterého je rozvětven. Tento krok předpokládá funkční a otestovaný nástroj s vytvořenou manuálovou stránkou. K tomu, pro budoucí udržovatelnost, je nutné přiložit skupinu testů, které mohou probíhat automaticky bez účasti vývojáře a vracejí binární výsledek pokud test uspěl nebo selhal. Více o publikaci nástroje je v kapitole 7.

Jednotlivé části implementace jsou detailněji popsány v následujících podkapitolách.

5.3 Realizace detekce chyb v deskriptoru

Detekce chyb v deskriptorech je implementována na úrovni porovnání kontrolních součtů a CRC. Další mechanismus, který je využíván, je ověření deklarovaného umístění deskriptoru vůči skutečnosti.

Kontrolní součty jsou implementovány mou funkcí

```
uint8_t calculate_checksum(descTag tag)
```

v souboru `udffsck.c`. Tato funkce pouze počítá kontrolní součet (viz. kapitola 4.1.1) nad zadaným tagem a výsledek výpočtu je vrácen jako návratová hodnota. Proto je zapouzdřena do funkce

```
int checksum(descTag tag)
```

která přidává porovnání s referenční hodnotou kontrolního součtu, který je uložen v tagu. Takto fukce vrací 0 při rozdílu a nenulovou hodnotu při shodě.

CRC je počítáno funkcí sdílené knihovny, která je součástí balíku. Ta byla opět zapouzdřena funkcí

```
int crc(void * desc, uint16_t size)
```

aby vracela tentokrát 0 při neshodě a nenulovou hodnotu při shodě. Oproti funkci pro výpočet kontrolního součtu je třeba funkci pro výpočet CRC dodat i délku. Důvodem tohoto oproti výpočetu kontrolního součtu pro *DescriptorTag* je, že *tag* má fixní délku a tu lze zjistit ze struktury, která jej popisuje pomocí funkce `sizeof()`. Deskriptory ovšem tuto vlastnost nemají. Jsou deskriptory, které mají délku fixní, ale část má délku proměnnou podle dat, která nesou (například délka struktury *File Entry* je závislá na tom, jestli se jedná o soubor nebo adresář a v případě adresáře ještě kolik dalších deskriptorů je uvnitř.) Stejně tak není možné počítat nad celou délkou sektoru, protože CRC se liší podle délky, i když je zbytek vstupní sekvence složen pouze z nulových hodnot. Více o tomto je popsáno v kapitole 4.1.2.

5.4 Oprava deskriptoru

Oprava deskriptoru při detekci chyby pomocí kontrolních mechanismů CRC, kontrolní součet nebo špatné pozice v *Tag Location* probíhá v šesti krocích. Předpokladem takové opravy je existence správného zdrojového deskriptoru nebo dostatek dat pro jeho znovuvytvoření. Kroky opravy jsou tyto:

1. Určení pozic zdrojového (správného) deskriptoru a cílového (defektního) deskriptoru.
2. Zkopírování zdrojového deskriptoru do paměti. Tímto vytvoříme nový cílový deskriptor.
3. Opravení položky *Tag Location* v tagu nového cílového deskriptoru podle pozice původního cílového deskriptoru.
4. Výpočet CRC nad novým cílovým deskriptorem a uložení výsledku do tagu. (Teoreticky nadbytečný krok, CRC by mělo být stejné jako u původního deskriptoru. Tento krok je užitečný pro vývoj kdy lze tímto ověřit, že byl zkorpírovaný celý zdrojový deskriptor.)
5. Výpočet kontrolního součtu tagu nového cílového deskriptoru a uložení výsledku do tagu. (Zde již je nutné kontrolní součet spočítat, protože se změnila pozice uložená v tagu podle nového umístění deskriptoru.)
6. Zkopírování nového cílového deskriptoru na místo původního cílového deskriptoru.

5.5 Mapa chyb

Účelem takzvané mapy chyb (neboli struktury `vds_sequence_t`) je zapouzdření chyb, které jsou detekovány napříč médiem v deskriptorech, vyjma chyb v deskriptorech dat. Její struktura je v tabulce 5.1.

Jak je vidět, je složena z malých struktur s názvem `metadata_t`. Ty jsou popsány tabulkou 5.2. Jak je vidět, obsahuje pouze identifikátor deskriptoru (použité pouze u VDS, více v kapitole 5.10), pozici deskriptoru a bitové pole chyb. Význam jednotlivých bitů je popsán v tabulce 5.3.

Adresa	Délka [B]	Jméno položky	Datový typ	Popis
0	21	anchor	pole metadata_t 5.2	Chyby v AVDP
21	56	main	pole metadata_t 5.2	Chyby v Hlavním VDS
77	56	reserve	pole metadata_t 5.2	Chyby v Záložním VDS
133	7	lvid	metadata_t 5.2	Chyby v LVID
140	7	pd	metadata_t 5.2	Chyby v SBD (referencováno z PD)

Tab. 5.1: Formát struktury `vds_sequence_t` (Mapa chyb)

Adresa	Délka [B]	Jméno položky	Datový typ	Popis
0	2	tagIdent	Uint16	Identifikátor deskriptoru
2	4	tagLocation	Uint32	Pozice deskriptoru
6	1	error	Uint8	Bitové pole nalezených chyb (tabulka 5.3)

Tab. 5.2: Formát struktury `metadata_t`

Název	Maska	Význam
`E_CHECKSUM`	`0b00000001`	Chyba v konrolním součtu
`E_CRC`	`0b00000010`	Chyba v CRC
`E_POSITION`	`0b00000100`	Chyba v umístění deskriptoru
`E_WRONGDESC`	`0b00001000`	Nenalezen očekávaný deskriptor (použito pouze pro AVDP)
`E_UUID`	`0b00010000`	Chyba UUID souboru
`E_TIMESTAMP`	`0b00100000`	Chyba časové značky souboru
`E_FREESPACE`	`0b01000000`	Chyba v objemu volného místa
`E_FILES`	`0b10000000`	Lišící se počet souborů nebo složek

Tab. 5.3: Chybové kódy pro strukturu `metadata_t`

5.6 Výstup a ovládání nástroje

Nástroj `udffsck` je navržen po vzoru ostatních nástrojů z rodiny `fsck`. Tomu bude odpovídat i jeho ovládání a výstup.

Vstupy různých `fsck` nástrojů se liší podle jednotlivých souborových systémů, ale kvůli zapouzdřitelnosti musí dodržovat návratové hodnoty pro různé druhy chyb. Jedná se o tyto hodnoty:

- 0 - Bez chyb
- 1 - Opraveny chyby na souborovém systému
- 2 - Opraveny chyby na souborovém systému, doporučený reboot (nepodporováno)
- 4 - Chyby souborového systému zůstaly neopraveny
- 8 - Chyba programu
- 16 - Chybné vstupní parametry
- 32 - Kontrola byla přerušena na základě uživatelského požadavku
- 128 - Chyba sdílené knihovny (nepodporováno)

Návratová hodnota je součtem těchto hodnot v případě souběhu.

Má implementace převzala tuto skladbu návratových hodnot s výjimkou kódů 2 a 128, které nejsou v tuto chvíli podporovány.

Protože se jedná o terminálový program, není obsaženo žádné grafické uživatelské rozhraní. Nástroj vypisuje informativní hlášení na standardní výstup `stdout` a chybová hlášení na chybový výstup `stderr`. Pokud je vyžadována interakce s uživatelem, jeho reakce je převzata ze standardního vstupu `stdin`.

Struktura volání nástroje `udffsck` je toto:

```
udffsck [-vvvciph] [-B BLOCKSIZE] medium
```

Jak je vidět, nástroj implementuje několik přepínačů a jeden vstupní parametr. Dále požaduje cestu k médiu, které má být kontrolováno (argument `medium`). Rozbor vstupních paramtrů je tento:

- `-B BLOCKSIZE` - Vnucení velikosti bloku místo autodetekce.
- `-c` - Médium bude pouze zkontrolováno, žádné chyby nebudou opraveny. Výchozí chování, pokud je nástroj zavolán bez přepínačů.
- `-i` - Interaktivní oprava za účasti uživatele. Při použití tohoto přepínače je před každou opravou požadována autorizace od uživatele.
- `-p` - Automatická oprava média. Výsledek této opravy je stejný jako kdyby uživatel v interaktivním režimu odpověděl na všechny otázky Ano.
- `-h` - Stručná nápověda k programu
- `-v` - Zvýšená úroveň výpisů do terminálu na hladinu *Warning*. K chybovým hlášením (které nelze potlačit) se přidají i varovná hlášení.

- `-vv` - Zvýšení úrovně hlášení na hladinu zpráv. Bude vypisováno to co na `-v` a k tomu pracovní zprávy. Vhodné pro uživatele.
- `-vvv` - Zvýšení úrovně hlášení na hladinu ladění. Bude vypisováno to co ne `-vv` a k tomu ladicí hlášení. Vhodné pro vývojáře.

Ukázkové volání nástroje může vypadat například takto:

```
udffsck -vvc -B 2048 /dev/sdb1
```

Stejné informace budou dostupné i s distribucí nástroje a to v manuálové stránce, kterou jsem napsal k `udffsck`. Její přepis je v příloze B. Ukázkový výstup programu je v příloze C.1 včetně komentáře jednotlivých částí výstupu.

5.7 Načtení média a jeho příprava na zpracování

Životní cyklus média v mé implementaci začíná předáním cesty k médiu při spuštění programu. Ta musí vést přímo k zařízení jako takovému, nikoli k místu, kam je připojeno.

Médium je na základě cesty načteno. Jednou pomocí funkce `open(2)` [22] pro práci samotnou, podruhé pomocí `fopen(3)` [23] pro nalezení velikosti média. Volání `fopen(3)` je pouze pro čtení, protože po úspěšném zjištění velikosti média není dále potřeba. Volání `open(2)` se liší podle dalších argumentů při spuštění programu. Pokud je médium pouze kontrolováno, je otevřeno pro čtení. Pokud je požadována i korekce (ať už automatická nebo interaktivní), je otevřeno pro čtení a zápis.

Dalším krokem po načtení je zamčení média pomocí funkce `flock(2)` [24]. Důvodem je zamezení paralelního přístupu k médiu při kontrole. Není žádoucí, aby médium mohlo být měněno během kontroly a korekce. Použití této funkce je implementováno jako neblokující. V případě nemožnosti uzamčít médium program skončí s chybou.

Po úspěšném zamčení média následuje namapování média do paměti funkcí `mmap(2)` [25]. Ta přebírá referenci na médium z funkce `open(2)` a stejně tak se liší její parametry podle požadavků na kontrolu nebo i korekci. Pokud je médium úspěšně namapováno do paměti, je možné přistoupit k práci s médiem jako takovým.

První otázka, která vyvstává, je duální otevření stejného souboru pouze pro zjištění velikosti média. Existuje funkce `fstat(2)` [26], která slouží ke zjištění údajů o souboru a pracuje s referencí z volání `open(2)`. Tato funkce naplní strukturu `struct stat` a ta obsahuje údaje jako je velikost souboru nebo ideální velikost bloku. Problém je, že tato funkce selže při použití se surovým médiem (tímto je myšlen přístup přímo na úložiště přes cestu `/dev/sdX`). Pokud by program pracoval pouze s obrazy disků, které jsou uložené jako soubor na souborovém systému, použití této funkce by bylo ideální. Ovšem v tomto případě je nejpřijatelnější přístup načtení média

znovu a použití funkce `fseeko(3)` [27] s parametrem `SEEK_END` a následně zjištění aktuální pozice funkcí `ftello(3)` [28]. Důvodem použití `fseeko(3)` a `ftello(3)` místo `fseek(3)` [29] a `ftell(3)` [30] je návratový typ `off_t`, který na rozdíl od `long` neumožňuje záporné hodnoty, tudíž je použití těchto funkcí preferované v nových projektech.

Druhá otázka vyvstává u použití funkce `mmap(2)` a výhodám oproti funkcím `read(2)` [31] a `write(2)` [32]. Důvodů je několik, počínaje pohodlností použití. Pokud je soubor mapovanán do paměti, je s ním z hlediska programu pracováno jako s jednorozměrným polem o velikosti mapovaného souboru. Tím pádem lze přistupovat na libovolné místo souboru bez nutnosti používat `lseek(2)`, lze snadno referencovat kusy souboru do struktur a poté s ním pracovat ve struktuře místo nutnosti ho do struktury kopírovat. Rizikem tohoto přístupu je setření hranice mezi médiem a operační pamětí a tudíž opatrnost při zápisu zpět do média, aby nedošlo k nežádoucí modifikaci souboru.

5.8 Kontrola přítomnosti identifikátoru souborového systému UDF

Po úspěšném načtení média je prvním krokem před jakoukoli další prací kontrola přítomnosti správného souborového systému, v tomto případě UDF. Tato kontrola probíhá kontrolou oblasti *Volume Recogniton Sequence* (VRS) pomocí funkce `is_udf()`. Tato funkce načte blok na sektoru 16. Tento blok je fixní místo na souborovém systému a je shodný pro UDF a ISO 9660, který je přímý předchůdce UDF. Zvláštností načítání VRS je velikost sektoru. Z důvodu zachování kompatibility s ISO 9660, který podporuje pouze velikost bloku 2048 B, je tato velikost převzata jako konstanta, nezávisle na skutečné velikosti bloku média. Ovšem i tento axiom má výjimku a to že toto platí pouze do velikosti bloku 2048 B. Na médiích s větší velikostí bloku je již použita skutečná velikost bloku místo 2048 B. To je možné díky skutečnosti, že ISO 9660 nikdy nebude možné použít na takovém médiu. U médií s menší velikostí bloku by ISO 9660 stále použít šlo, stačí pouze zapisovat na více bloků místo jednoho při zápisu 2048 B.

Kontrola samotná probíhá načtením struktur VRS, kde se vyhledávají deskriptory popisující UDF, t.j. deskriptory obsahující identifikátor `NSR01`, `NSR02` nebo `NSR03`. Pokud je takovýto identifikátor nalezen, a spolu s ním je i nalezen identifikátor `TEA01` (*Terminating Extended Area*), je médium prohlášeno za správné a může se přistoupit ke kontrole. Pokud je nalezena identifikace UDF ale ne kompletní *Extended Area*, může to znamenat neuzavřené médium a to pro další kontrolu znamená jiné umístění AVDP. Kontrola je ale možná. Pokud nebyl nalezen identifikátor UDF,

kontrola končí chybou a program končí, protože nebylo možné detekovat přítomnost UDF.

Druhou činností, kterou funkce `is_udf()`, pokrývá je první pokus o detekci velikosti sektoru. Ta postupně zkouší načíst platné deskriptory pro různé povolené velikosti sektoru (hodnoty dělitelné beze zbytku 512, v praxi ovšem pouze omezená řada 512, 1024, 2048, 4096 B) a ukládá hodnotu, kdy se to podaří. Prakticky by se to mělo podařit pouze pro velikost sektoru 2048 B a 4096 B. Pro velikosti sektoru média 2048 B včetně je obvykle detekovaná velikost sektoru 512 B (neboli nejnižší možná). S touto hodnotou poté dále pracuje detekce AVDP, která ji používá jako výchozí bodu svého hledání velikosti sektoru.

5.9 Načtení, kontrola a korekce deskriptoru AVDP

Anchor Volume Descriptor Pointer (AVDP) je klíčovou částí UDF, protože určuje výchozí bod pro čtení souborového systému (na rozdíl od VRS, které pouze určuje přítomnost UDF). Jak je uvedeno v kapitole 2.3.2, UDF rozlišuje čtyři možná umístění AVDP, přičemž přítomné na mediu může být jedno nebo až tři.

Jejich načítání pro potřeby další práce se souborovým systém lze popsat algoritmem na obrázku 5.3. Algoritmus prochází možné lokace AVDP, dokud neuspěje nebo nedojdou možnosti. Pro potřeby kontroly a opravy média je třeba po nalezení alespoň jednoho správného AVDP ostatní podle něj opravit.

Jak je uvedeno, je možné, aby bylo na médiu AVDP uloženo jednou nebo až třikrát. Toto souvisí se stavem finalizace média. Více o finalizaci je v kapitole 2.2.

Z toho vyplývá, že oprava AVDP je možná pouze na uzavřených médiích. Důvodem je požadavek opravného algoritmu na přítomnost alespoň jednoho nepoškozeného (správného) deskriptoru, který bude použit jako vzor pro opravu. Otevřená (neukončená) média mají pouze jeden AVDP, proto v případě jeho poškození není oprava možná.

Jedinou variantou by v takovém případě bylo prohledávání média po sektorech a hledání VDS deskriptorů. Toto ovšem není v tuto chvíli implementováno.

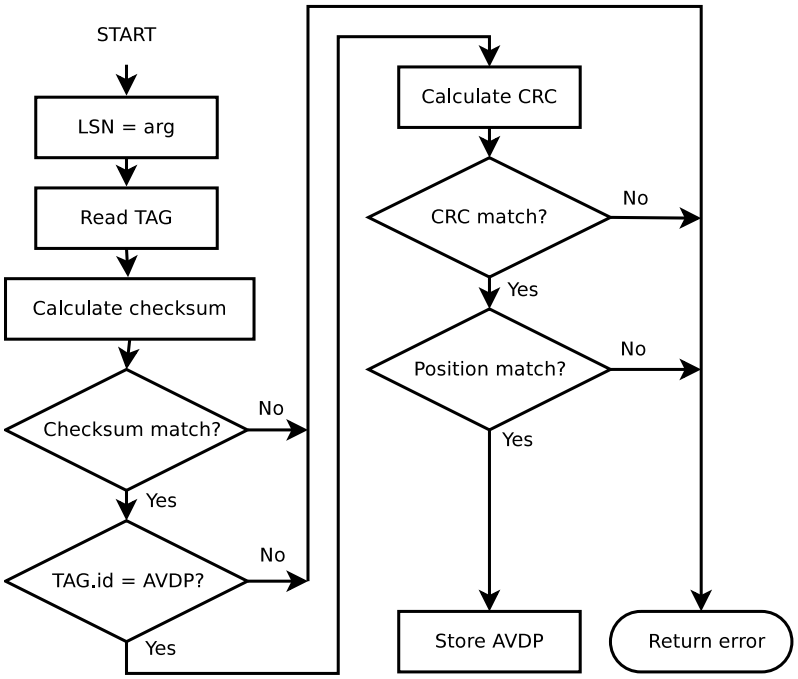
Poté, co jsou nalezeny AVDP na svých pozicích (t.j. sektoru 256 a na posledním sektoru, případně na sektoru 512) a je určeno, který z nich je poškozený a který je v pořádku, lze přistoupit k opravě, pokud je alespoň jeden v pořádku.

Samotný opravný algoritmus je popsán v kapitole 5.4.

Detekovatelnou a opravitelnou chybou je nedodržení specifikace UDF[2], kapitola 2.2.3.1 s 2.2.3.2 týkající se rezervované délky pro VDS. Minimální délka této oblasti je 16 sektorů pro hlavní VDS a 16 sektorů pro záložní VDS. Pokud je tento konflikt

nalezen, oprava spočívá v kontrole možnosti rozšíření a následné nastavení správné velikosti rezervovaného místa.

Případné nalezené chyby jsou uloženy do struktur popsaných v kapitole 5.5. V tomto případě může dojít k chybám v kontrolním součtu, v CRC, poloze deskriptoru a nalezení platného *jiného* deskriptoru na pozici vyhrazené pro AVDP.



Obr. 5.3: Zjednodušený algoritmus načtení AVDP

5.10 Načtení, kontrola a korekce deskriptoru VDS

Po úspěšném nalezení funkčního AVDP je možné načíst *Volume Descriptor Sequence* (VDS). Tato skupina metadat je uložena na dvou místech, typicky jedno na začátku a druhé na konci média (ale není to pravidlem). Pozice obou je uložena právě v AVDP.

Tato skupina deskriptorů je popsána v kapitole 2.3.3. Deskriptory jsou uloženy za sebou v jednotlivých sektorech, přičemž poslední musí být vždy *Terminating Descriptor*. Jak hlavní, tak záložní VDS by měly být ekvivalentní (ačkoli ne nutně ve stejném pořadí.)

Jednotlivé deskriptory jsou načítány a ukládány do struktur pro další zpracování. První kontrola během načítání spočívá v porovnání kontrolního součtu tagu deskriptoru (viz kapitola 4.1). Pokud tato kontrola uspěje, je možné provést porovnání CRC, protože již máme díky ověřenému tagu jistou totožnost deskriptoru, jeho předpokládané CRC a polohu. Pokud kontrola CRC uspěje, zkontroluje se poloha

deskriptoru vůči deklarovanému umístění v tagu. Pokud i tato kontrola je v pořádku, lze prohlásit tag za správný. Pokud by byl v pořádku jak kontrolní součet, tak CRC, ale bylo by špatně deklarované umístění, lze s tagem pracovat, protože data která nese, jsou s největší pravděpodobností v pořádku, ale je nutné opravit jeho umístění a následně přepočítat kontrolní součet a CRC. Oprava VDS je podobná jako oprava AVDP. Předpokladem úspěšné opravy je opět existence alespoň jednoho správného zdrojového deskriptoru od každého typu, který ve VDS je. Pokud je toto splněno, je oprava možná.

Algoritmus je navržen takto:

1. Během načítání VDS (jak hlavního, tak záložního) je vytvářena mapa chyb na jednotlivých deskriptorech. Formát mapy je popsán v tabulce 5.1.
2. Na základě údajů mapy chyb je přistoupeno k opravě. Pokud je alespoň jeden z deskriptorů v pořádku, je použit algoritmus z kapitoly 5.4. Pokud ne, je vypsána chyba a pokračuje se k dalšímu deskriptoru.
3. Krok 2 se opakuje, dokud není nalezen *Terminating Descriptor*. Poté je oprava ukončena, protože byl nalezen konec VDS.

Právě u korekce VDS může dojít k situaci, kdy bude část opravena a část nikoli. Dalším důležitým limitem je předpoklad, že jak hlavní, tak rezervní VDS mají stejné pořadí deskriptorů. Je to konvence, ovšem nikoli standard.

5.11 Načtení, kontrola a korekce deskriptoru LVID

Logical Volume Integrity Descriptor (LVID) je důležitou částí UDF. Je umístěné mimo VDS a jeho pozice je uložena v LVD pod položkou *Integrity Sequence Extent*. LVID není redundantní, což zjednodušuje práci s ním, ale zároveň se kvůli tomu jedná o slabé místo UDF. Ovšem je potřeba vzít v potaz skutečnost, že LVID lze v případě nepoškozeného souborového systému vytvořit znovu zpětně. V případě poškozeného souborového systému jej lze vytvořit též, ale s rizikem ztráty dat z důvodu nekompletní informace o souborovém systému v jiných deskriptorech.

Po kontrole kontrolního součtu a CRC je přistoupeno k samotnému načtení dat. Pro snažší práci s daty uloženými v LVID jsou vybraná data ukládána do struktury `filesystemStats`.

Protože nadřazeným standardem UDF je ECMA-167, je LVID popsáno podle ní. Ovšem samotnou implementací je UDF standard a ten využívá pole *Implementation Use*, kam ukládá tyto údaje:

- Počet souborů na svazku
- Počet adresářů na svazku
- Minimální revize UDF, se kterou je možné médium číst

- Minimální revize UDF, se kterou je možné na médium zapisovat
- Maximální revize UDF, se kterou je možné na médium zapisovat

Podobná situace je u položky *Logical Volume Contents Use*. UDF tuto položku používá k uložení *Logical Volume Header Descriptor*, což je struktura popsaná pomocí standardu ECMA-167 s jediným cílem a tím je uložení *Unique ID*, neboli unikátního identifikátoru, který nese každý soubor a adresář na médiu. Význam *Unique ID* je popsán v kapitole 5.13. LVID zde nese informaci o tom, jaké *Unique ID* bude použito pro další záznam dat (což zároveň znamená, že toto *Unique ID* musí být nejvyšší ze všech, co jsou uloženy na médiu.)

Další částí je *Free Space Table*, což je tabulka obsahující informaci o velikosti logického svazku a volného místa na něm. Oba údaje jsou v logických blocích, nikoli v bytech.

Důležitou částí je i časová značka poslední modifikace logického svazku. Tato značka, uložená v UTC, by měla být vždy nejnovější ze všech časových značek na svazku.

Poslední položkou, která je důležitá pro prácí se svazkem, je *Integrity Type*. Ten může být buď otevřený (*Open*) nebo uzavřený (*Close*). Princip otevřeného a uzavřeného média je popsán v kapitole 4.2. Pokud je uzavřený, tak by měl být svazek v konzistentním stavu. Jestliže je otevřený, je možné, že probíhá zápisová operace, nebo že byl svazek během ní nesprávně odebrán. Tento indikátor pomáhá vyvolat požadavek na kontrolu média v operačním systému a pro kontrolu to je důležitý ukazatel upozorňující na pravděpodobnou nesrovnalost mezi deklarovaným volným místem a skutečností.

Korekce LVID je odlišná od ostatních korekcí z důvodu absence redundantního LVID. Z toho vyplývá, že LVID nelze v případě poškození opravit (t.j. pokud selže kontrolní součet nebo CRC.) LVID ale obsahuje důležité informace o konzistenci média (viz kapitola 2.3.4) a proto je korekce soustředěna na ně.

Opravují se tyto položky:

- Vyvolání opravy SBD (kapitola 5.12)
- Počet souborů a složek.
- Následující *Unique ID*.
- Datum posledního záznamu se nastaví na čas opravy.
- Opraví se tabulka volného místa *Free Space Table*.
- Médium se uzavře (*Integrity type* se nastaví na *Closed*).
- Přepočítá se kontrolní součet a CRC.

Korekce je spouštěna z několika různých důvodů, které sice ve většině případů nastávají současně, ale mohou se vyskytnout i odděleně. Jedná se o tyto zdroje:

- Pokud bylo nalezeno *Unique ID* stejné nebo větší jako uloženo v LVID.

- Pokud byl nalezen soubor s novějším datem, než je datum posledního záznamu v LVID.
- Pokud byla nalezena nesrovnalost v počtu souborů nebo složek uložených v LVID vůči skutečnému počtu.
- Pokud nekoresponduje objem volného místa s hodnotou deklarovanou v *Free Space Table*.
- Pokud byl *Integrity Type* ponechán ve stavu *Open*.

5.12 Načtení, kontrola a korekce deskriptoru SBD

Space Bitmap Descriptor je deskriptor popisující volné a obsazené místo na logickém svazku z hlediska umístění jednotlivých bloků. Jedná se o vektor bytů (*Bitmap*) o délce $PocetLBN/8$. Spolu s ním je uložena jeho délka (*Number of Bytes*) a skutečný počet bloků (*Number of Bits*). SBD je popsáno v kapitole 2.3.3.

Každý jeden byte v tomto vektoru odpovídá osmi logickým blokům v přirozeném pořadí (nejméně významný bit odpovídá prvnímu bloku, nejvíce významný bit odpovídá osmému bloku).

Hodnota bitu označuje stav bloku. Logická 1 odpovídá volnému bloku, logická 0 obsazenému bloku. Přebývající bity v posledním bytu se označují jako obsazené.

Korekce probíhá tak, že během průchodu souborovým stromem (kapitola 5.13) se vytváří nový vektor obsazeného místa o stejné velikosti jako původní. Ten slouží jako referenční mapa, která bude nakopírována na místo původní v případě nutnosti opravy.

Oprava je vyvolána automaticky při opravě LVID (kapitola 5.11), při rozdílu počtu použitých bloků oproti počtu ve vektoru *Bitmap* nebo pokud při načítání došlo k chybě v kontrolním součtu nebo CRC.

Nutno podotknout, že SBD nemusí být vůbec na médiu přítomno. Jeho určením je totiž přednostně optimalizace zápisů u *Rewritable* médií z hlediska přepisu jednou zapsaných sektorů. To ovšem nedává smysl pro *Write Once* a *Read Only* média, proto je tento deskriptor u nich vynechán.

5.13 Kontrola a obnova stromu souborů

Obnova stromu uživatelských dat je klíčová část programu. Bez ní nemá tento nástroj význam. Vzhledem k faktu, že `fsck` nemá za cíl detekovat chyby v datech samotných a UDF také neimplementuje ochranné algoritmy na data samotná ale jen na metadata, není bohužel možné poškozená data ani detekovat a tudíž ani

opravit. Je možné se pokusit zrekonstruovat adresářovou strukturu a určit místo, kde by data měla být.

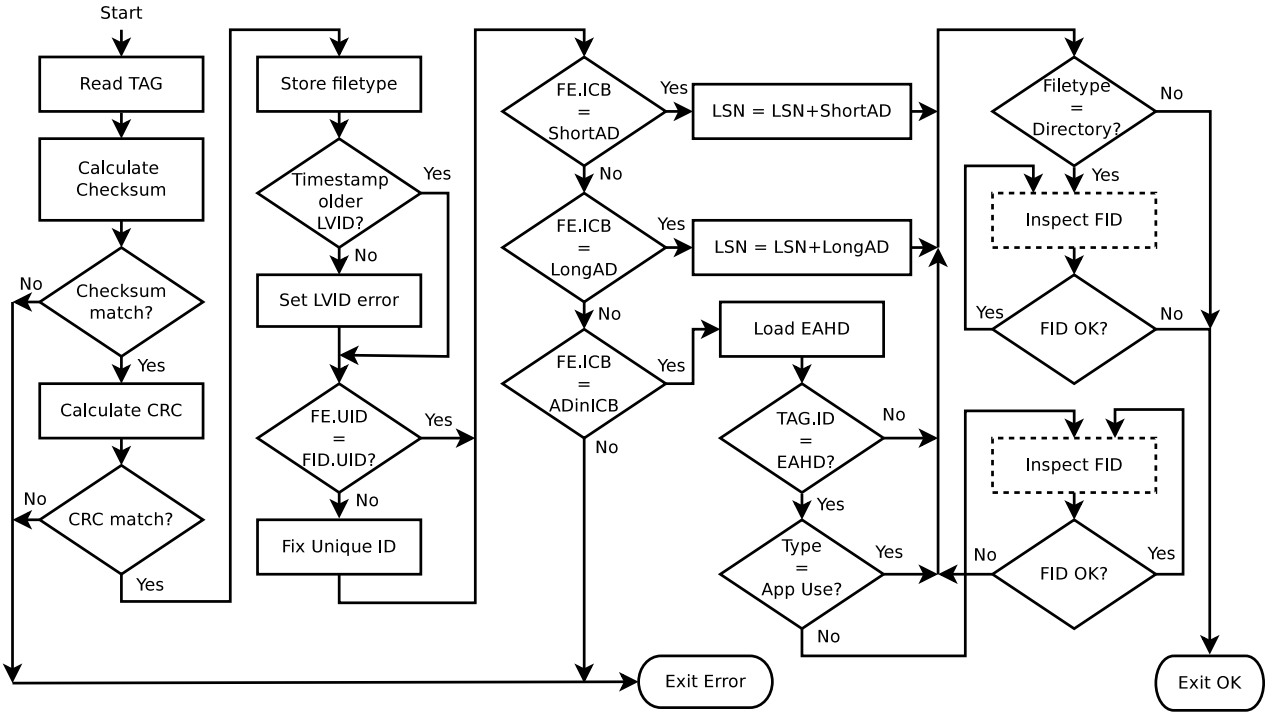
Důvodem vynechání dat z detekce a korekce na úrovni `fsck` je skutečnost, že fyzické médium má integrované ochranné mechanismy v sobě, například v podobě ECC bloků. Proto se předpokládá, že pokud jsou data jednou korektně uložena, médium se postará o jejich správnost.

Ústředními deskriptory u stromu souborů jsou deskriptory *(Extended) File Entry* (FE/EFE), který je popsaný v kapitole 2.4.2 a *File Identifier Descriptor* (FID), který je popsaný v kapitole 2.4.3.

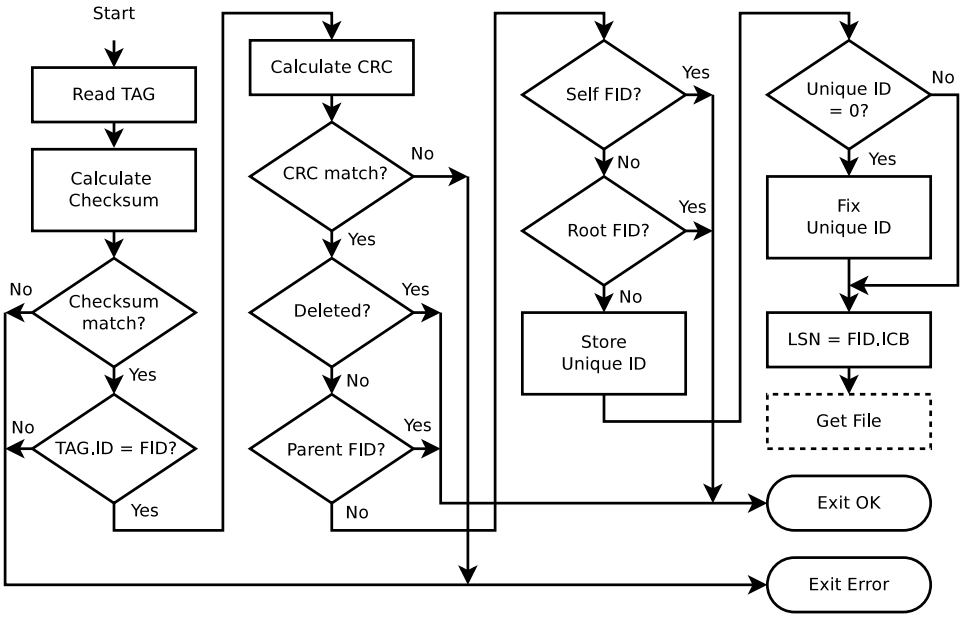
Algoritmus načtení adresářové struktury je zachycen na obrázku 5.4. Tento algoritmus začíná po úspěšném načtení FSD, odkud je získána pozice FE kořenového adresáře, která je předána algoritmu. Od tohoto bodu probíhá algoritmus zachycený na obrázku. Předpokladem úspěšného průchodu je správnost všech kontrolních mechanismů. Ty není možné opravit, pouze je možné detekovat chybu. Ovšem pokud je deskriptor identifikován a selhala kontrola CRC, je uživateli v interaktivním režimu poskytnuta možnost tuto chybu ignorovat. Opravitelné chyby jsou chyby týkající se *Unique ID* a časových značek. Po fázi detekce chyb následuje fáze hledání následující části stromu. Existuje několik druhů alokačních deskriptorů. Liší se způsobem jakým udržují informaci a kam mohou ukazovat. Povolené varianty jsou *Long Allocation Descriptor* (LongAD), který ukazuje na logický blok v libovolném oddílu. *Short Allocation Descriptor* (ShortAD) naopak může ukazovat na logický blok pouze ve svém oddílu. Poslední variantou je uložení deskriptorů FID přímo v FE (varianta ADinICB). Tato větev využívá volného místa, které vzniká za většinou deskriptorů, takže jsou FID uloženy přímo. Je zde použit deskriptor *Extended Attribute Header Descriptor*, který v sobě nese buď nějaká implemenetační data, nebo sekvenci deskriptorů FID. Deskriptory FID jsou v obou případech procházeny iterativně dokud neskončí jeho načtení chybou (ať už protože je poškozený, tudíž není možné strom dále projít, nebo proto že sekvence skončila.) Zpracování samotných deskriptorů FID je popsáno ve vlastním algoritmu, který je na obrázku 5.5.

Algoritmus pro načítání deskriptorů FID je přímější. Začátek algoritmu je stejný jako v předchozím případě. Poté následuje sekvence dotazů, kdy nemá význam FID sledovat. Jedná se o příznak smazání, referenci na rodičovský adresář, sebe sama a kořenový adresář. Vzhledem k faktu, že chceme postupovat do hloubky, tyto reference nás nezajímají a proto algoritmus v tomto bodě končí (t.j. v algoritmu na obrázku 5.4 se přejde ke kontrole dalšího FID). Pokud touto částí algoritmus projde, přistoupí se k uložení aktuálního *Unique ID* pro porovnání s FE a jeho kontrola na nulovost. Ta je případně opravena. Poté už následuje načtení pozice nového deskriptoru FE a jeho zpracování. Tím se algoritmus vrací opět k předchozímu algoritmu na obrázku 5.4.

Během průchodu se sleduje několik parametrů, které jsou nutné ať už pro kontrolu a opravu stromu nebo pro opravu metadat v LVID 5.11 a SBD 5.12. Jedná se o tyto:



Obr. 5.4: Algoritmus načítání descriptorů FE (funkce *Get File*). Tečkovaně naznačený blok odkazuje na obrázek 5.5.



Obr. 5.5: Algoritmus načítání descriptorů FID (funkce *Inpspect FID*). Tečkovaně naznačený blok odkazuje na obrázek 5.4.

- Je *Unique ID* nenulové (nulové musí být pouze pro kořenový adresář)?
- Je *Unique ID* shodné pro FE a FID?
- Je *Unique ID* menší než hodnota uložená v LVID?
- Jaký je čas poslední změny? Je tento čas později než je čas uložený v LVID?

Pokud je kterýkoli dotaz zodpověděn záporně, je nastaven příznak chyby.

Další parametr, který sice není zachycen pro zachování jednoduchosti vývojového diagramu, je *Tag Serial Number*. Jedná se o parametr, který slouží k opravě v případě katastrofické poruchy souborového systému a bylo by nutné jej zrekonstruovat. Je uložen v každém tagu souborového systému a pokud je tato možnost povolena (nemusí být), mají všechny stejnou hodnotu, která musí být 2 nebo větší. Tento údaj slouží k přiřazení daného tagu (a potažmo deskriptoru) ke správnému deskriptoru AVDP. Tím je vytvořen seznam deskriptorů, který je již možné rekonstruovat. Celý mechanismus je popsán ve specifikaci UDF [2], kapitola 2.1.6.

Několikrát zmiňovaný je parametr *Unique ID*. Jedná se o příznak, který je unikátní pro každý soubor nebo adresář. Je stejný pro FE a FID. Důvodem je šance na obnovu v případně katastrofického poškození souborového stromu. Pokud by se muselo přistoupit ke hledání jednotlivých FE ve všech sektorech logického svazku a párovat je s FID v adresářích, toto by byla jediná šance na obnovu.

Základním předpokladem pro použití tohoto algoritmu je ztráta některého z adresářů (například kořenového adresáře). To způsobí nemožnost načíst strom souborů z důvodu chybějícího vstupního bodu do stromu souborů. Nutno podotknout, že tento algoritmus není zahrnut v implementaci, ale jeho myšlenková struktura je následující:

1. Díky známé velikosti logického bloku a známému umístění deskriptorů v nich (vždy na jeho začátku) lze procházet logickým svazkem po blocích a na každém se pokusit načíst *File Entry* (FE) nebo *Extended File Entry* (EFE), verifikovat jeho platnost pomocí jeho polohy, CRC a kontrolního součtu.
2. Takto nalezený seznam FE a EFE bude vytvářet samostatné podstromy souborů (pokud médium obsahovalo další adresáře) a zbylé soubory bez rodičovského adresáře. Tato část probíhá podobně jako procházení stromem souborů, zkráceně takto:
   (a) Každý nalezený FE nebo EFE je klasifikován jako soubor nebo adresář. Pokud se jedná o adresář, je prohledán pro přítomnost *File Identifier Descriptor* (FID). Pokud jsou nalezeny, dojde k pokusu spárovat je podle *Unique ID* s ostatními nalezenými FE a EFE. Každý takto spárovaný FE (EFE) je vyřazen ze seznamu nalezených FE (EFE) a je zařazen do podstromu. Pokud se jedná o soubor, neděje se nic a pokračuje s k dalšímu nalezenému FE (EFE).
   (b) Takto se vytvoří struktura podstromů a zbytku nezařazených FE a EFE.

3. Nyní máme části původního stromu a zbytek FE (EFE). Ať už kořeny jednotlivých podstromů (nejvyšší adresář) nebo osamocené FE (EFE). Ty už není možné dále zařadit, proto budou zařazeny do chybějícího adresáře, který bude znovu vytvořen na místě původního. Jsou zde ovšem jistá omezení. Obnova není schopná zpětně získat názvy jednotlivých FE (EFE), protože ty jsou uloženy ve FID rodičovského adresáře. Stejně tak pokud by bylo nenávratně ztraceno více adresářů, nebylo by možné rozlišit, které FE (EFE) patří do kterého.

Jak je vidět ve výše popsaném algoritmu, obnova není absolutní, ale může poskytnout dobrý výchozí bod pro záchranu dat.

Důležitou částí, kterou je nutno popsat, je způsob vyřešení nedokončeného zápisu a způsoby jeho detekce. Vzhledem k faktu, že deskriptor LVID 2.3.4 pomocí položky *Integrity Type* pouze poukzauje, že médium nebylo korektně odebráno a ovladač může zaznamenat potřebné údaje o souboru dříve, než začne ukládat data, je potřeba mít způsob detekce této situace.

V tomto případě spoléháme na nekonzistentní deklarovaný objem uložených dat (položka deskriptoru FE *Information Length*) a informaci o objemu skutečně uložených dat (položka deskriptoru FE *Logical Blocks Recorded*), která by měla být aktualizována průběžně během, případně na konci zápisu. Ve chvíli, kdy se tyto údaje liší, soubor není kompletní.

V mé implementaci jsem zaujal přístup smazání daného souboru vzhledem k faktu, že informace kterou nese není kompletní. To je zajištěno označením daného souborou příznakem *Deleted* v položce *File Characteristics* v deskriptoru FID. Druhý úkol který je třeba v deskriptoru FID udělat je dle standardu ECMA-167[4], revize 3, kapitola 4/14.4.5, vynulování položky ICB, čímž se zajistí zničení reference pokud by při čtení nebyl kontrolován příznak smazání. Pro kompletnost byl vynulován celý deskriptor FE defektního souboru aby při případné obnově z katastrofického poškození nebyl mylně vyhodnocen jako platný. Takto vyřešený nedokončený zápis je viditelný při analýze souborového systému jako smazaný soubor a zároveň uživatel nenabývá mylného dojmu existence souboru, což byl efekt, který je patrný při použití nástroje v prostředí Microsoft Windows (nástroj `CHKDSK`).

6 VÝSLEDKY A TESTOVÁNÍ NÁSTROJE PRO DETEKCI A KOREKCI CHYB

V této kapitole jsou popsána testovací data, metodika testování a dosažené výsledky. Kapitola je koncipována informativně a pracuje s vyběrovou množinou médií, která neobsahuje všechny možné varianty vzhledem k vysoké variabilitě UDF. Byl zvolen přístup otestování vůči nejběžnějším variantám médií, což bývá výchozí nastavení nástrojů pro vytvoření UDF, obvykle se lišící pouze nosným médiem, velikostí bloku a verzí UDF. Toto bylo ověřeno vůči médiím, které vznikly na platformách GNU/Linux, Microsoft Windows a Apple macOS.

6.1 Testovací data

Testovacími daty jsou myšlena různá média, která jsou naformátována souborovým systémem UDF. Tato média různých parametrů jsou záměrně poškozována a následně analyzována.

Testovací sada obsahuje média všech povolených velikostí sektorů (t.j od 512 B do 4096 B) a různých verzí (od 1.02 do 2.60 přičemž většina testů je mířena proti 2.01.) Poškozování médií probíhalo dvěma způsoby: synteticky a nahodile.

Syntetické poškození média probíhalo ručním přepsáním některých vybraných částí média pro vytvoření chyby v daném deskriptoru. Takto byly testovány kontrolní součty a CRC a posléze opravy takto poškozených deskriptorů. Nahodilé poškození média bylo vytvořeno záměrným přerušením zápisu na médium odebráním zařízení. Nahodilé proto, že není předem jasné, co bude poškozeno a jak. Tyto testy cílí na obnovu stromu souborů a opravu a uzavření LVID.

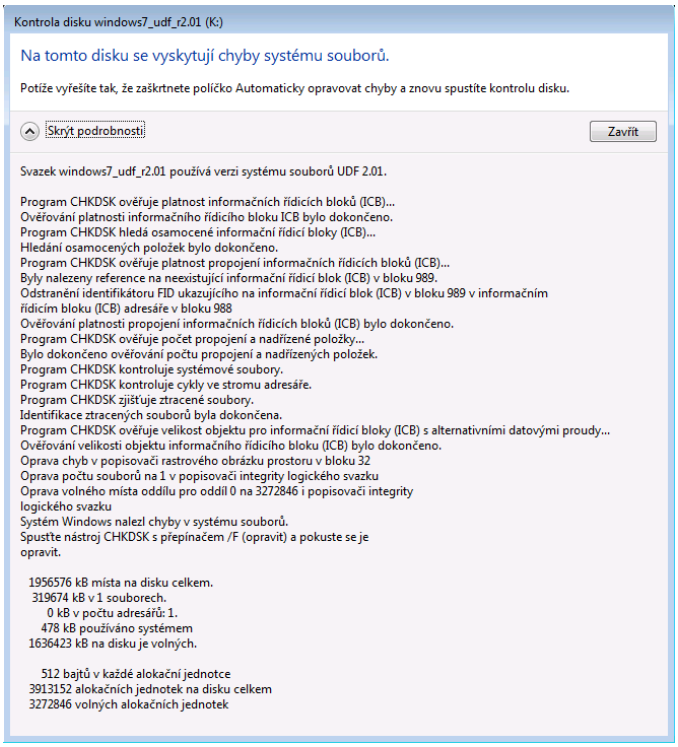
Tímto způsobem bylo vytvořeno přes 20 GB testovacích dat ve 32 vzorcích.

Pro zjištění chyb na médiích a pro zjištění úspěšnosti detekce a následné korekce byl použit nástroj společnosti Philips `udfct` [7] určený ke kontrole dodržování standardu UDF.

6.2 Metodika testování

Metodika testování je navržena následovně:

1. Vytvoří se čerstvá kopie testovacích dat.
2. Spustí se referenční nástroj `udfct` [7] pro zjištění aktuálního stavu média pro následné srovnání.
3. Spustí se nástroj `udffsck` v kontrolním režimu a porovnají se výstupy s referencí.



Obr. 6.1: Výsledek kontroly poškozeného média nástrojem `CHKDSK`

4. Pokud médium obsahuje chyby, spustí se nástroj `udffsck` v opravném režimu.
5. Nástroj `udffsck` je spuštěn opět v kontrolním režimu. Pokud došlo v předchozím kroku k plné opravě (návratová hodnota 1), mělo by médium být nyní bez chyb (návratová hodnota 0)
6. Spustí se referenční nástroj `udfct` a opět se porovnají výstupy. Pokud byla oprava úspěšná, médium by nemělo nyní vykazovat chyby, nebo pokud chyby vykazuje, neměly by ovlivňovat funkci souborového systému.

Tato metodika byla navržena pro potřeby vývoje nástroje a pro zajištění maximálního možného pokrytí případných chyb média (obzvláště u nahodile poškozených médií.) Zjednodušená verze tohoto testu byla implementována pro potřeby automatického testování službou Travis CI. Ta je omezena pouze na testování vlastního nástroje bez reference a testuje vůči předpokládaným návratovým hodnotám. V principu jde o vynechání kroků 2 a 6 z předchozí sekvence.

Tento přístup pomohl odhalit řadu potíží během vývoje, kdy opravením jedné chyby byla vytvořena omylem chyba jiná. Takto byla chyba podchycena okamžitě po vzniku, takže mohla být okamžitě vyřešena bez opakované analýzy problému.

6.3 Srovnání s ostatními řešeními

Pro ověření správné funkce nástroje byla vybraná média otestována i na jiných operačních systémech pro srovnání funkce. Jako ukázku tohoto přístupu lze zvolit médium, které vzniklo na MS Windows pro flash disk (tedy *Overwritable* médium) a následně bylo záměrně poškozeno přerušením zápisu.

Byla provedena kontrola disku na MS Windows 7 pomocí nástroje *CHKDSK*. Výsledek jeho kontroly před jakoukoli opravou je na obrázku 6.1. Kontrola byla provedena i na MS Windows 10 se stejným výsledkem. Podle nástroje `CHKDSK` je médium opravitelně poškozeno.

Podobný výsledek byl získán v Apple macOS programem `fsck_udf`. Přepis jeho výstupu je ve výpisu v příloze D.4. Z obou výstupů je vidět, že oba nástroje došly ke stejným poruchám na médiu. Médium bylo v této fázi zazálohováno pro srovnání různých variant. První varianta je oprava pomocí mého nástroje `udffsck`. Druhá varianta je oprava pomocí nástroje MS `CHKDSK`. Oprava pomocí `fsck_udf` z Apple macOS není možná, protože nástroj nemá implementovány žádné opravné algoritmy. Pro referenční srovnání je přiložen výstup z programu `udfct` společnosti Philips, jehož výstup je vzhledem k jeho délce v příloze D.1.

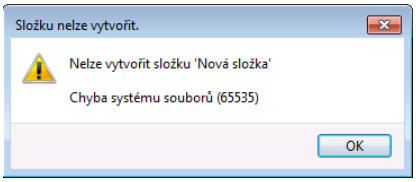
Výpis 6.1: Výsledek opravy poškozeného média programem `udffsck`

```
1  Verbosity increased to WARNING.
2  Verbosity increased to MESSAGE.
3  We try to fix medium automaticaly.
4  Medium to analyze: /dev/sdb
5  AVDP[0] successfully loaded.
6  AVDP[1] successfully loaded.
7  AVDP[2] successfully loaded.
8  Sectorsize: 512
9
10 Medium file tree
11 ----------------
12 .....:..rwx.:.rwx.:.rwx DIR     2017-05-05 15:58:54.804378+00:00       152  <ROOT>
13   .....:..rwx.:.rwx.:.rwx FILE    2009-11-12 19:46:14.000000+00:00  327345425  "gtd
       .mp4"
14 [ERROR] (2017-01-16-191632.webm) Tag Serial Number differs.
15 (2017-01-16-191632.webm) Tag Serial Number was fixed.
16 [ERROR] (2017-01-16-191632.webm) File size mismatch. Probably unfinished file write.
17 Removing unfinished file...
18 (2017-01-16-191632.webm) Unifinished file was removed.
19
20 Filesystem status
21 ------------------
22 Volume set identifier:113A1E01 UDF Volume Set
23 Partition identifier:
24 Next UniqueID: 18
25 Max found UniqueID: 17
26 Last LVID recoreded change: 2017-05-05 16:15:01.278032+00:00
27 expected number of files: 2
28 expected number of dirs:  1
```

```
29 counted number of files: 1
30 counted number of dirs:  1
31 UDF rev: min read:  0201
32          min write: 0201
33          max write: 0201
34 Used Space: 327836672 (640306)
35 Free Space: 1665865216 (3253643)
36 Partition size: 2003533824 (3913152)
37 Expected Used Space: 337668608 (659509)
38 Expected Used Blocks: 882490
39 Expected Unused Blocks: 3030662
40 [ERROR] 19203 blocks is unused but not marked as unallocated in Free Space Table.
41 [ERROR] Correct free space: 3272846
42 [ERROR] 242184 blocks is unused but not marked as unallocated in SBD.
43
44 VDS verification status
45 ------------------------
46 [0] PVD is fine. No fixing needed.
47 [1] PD is fine. No fixing needed.
48 [2] LVD is fine. No fixing needed.
49 [3] USD is fine. No fixing needed.
50 [4] IUVD is fine. No fixing needed.
51 [5] TD is fine. No fixing needed.
52 [ERROR] Opened integrity type. Some writes may be unfinished.
53 [ERROR] Number of files or directories is not corresponding to counted number
54 [ERROR] Free Space table is not corresponding to reality.
55 PD SBD recovery was successful.
56 LVID recovery was successful.
57 All done
```

Výsledek z nástroje `udffsck` je vidět na výpisu 6.1. Jak je vidět, byla nalezena řada chyb a všechny byly opraveny. Funkčnost souborového systému byla posléze ověřena připojením k MS Windows 7, kde na něj byla úspěšně nakopírována data. Stejně tak výstup z programu MS `CHKDSK` nedetekoval žádné problémy. Nabízí se otázka, proč nebyla funkce souborového systému ověřena i na OS Linux. Důvodem je benevolence implementace ovladače v OS Linux, který dokáže bez potíží zapisovat i na defektní souborový systém. Problém je, že ho tím dále poškozuje. Proto byl jako kontrolní test zvolen MS Windows 7, který přistupuje k připojení souborového systému striktněji.

Nyní pokud se vrátíme k zálohované variantě a zkusíme pro srovnání provést obnovu pomocí MS `CHKDSK`. Na konci opravy, kdy program hlásí, že opravil všechny



Obr. 6.2: Chyba při pokusu o zápis na médium opravené nástrojem `CHKDSK`

nalezené chyby, zkusíme taktéž médium připojit a nahrát data. Nyní ale dochází k selhání, viz obrázek 6.2. Opětovná kontrola pomocí `CHKDSK` nenalezne žádné chyby. Verdikt je, že médium je nenávratně poškozeno, ačkoli `CHKDSK` nenalézá chybu. Po kontrole referenčním nástrojem `udfct`, jehož výstup je opět v příloze D.2, je vidět, že program `CHKDSK` sice opravil nalezené chyby, ale také vytvořil nové. K jeho opravě by bylo nutné částečné přeskupení struktur na médiu, což ostatně svým během udělal a právě tím médium poškodil, protože tak neprovedl v souladu se specifikací. Pokud použijeme můj nástroj `udffsck`, zjistíme, že některé z chyb jsou opravitelné, ale jejich odstranění vyvolává další, které můj nástroj již bohužel nedetekuje. Jejich nález je opět díky `udfct`, jehož výpis je v příloze D.3. Ovšem na rozdíl od chyb, které vytvořil program `CHKDSK`, nemají mnou vytvořené chyby dopad na funkci souborového systému, to dokazuje i opětovný pokus o připojení a nahrávání dat na MS Windows 7, který nyní neselže.

Cílem této analýzy bylo ukázat, že nástroj `udffsck` dokáže opravit poškozený souborový systém UDF tak, že i striktní ovladač MS Windows dokáže bez potíží s médiem pracovat na rozdíl od jejich vlastního řešení `CHKDSK`. Vzhledem k faktu, že neexistuje jiné řešení, ať už otevřené nebo uzavřené, schopné jak detekce chyb, tak jejich korekce pro souborový systém UDF, považuji tento výsledek za úspěch.

6.4 Dosažené výsledky

Byl vytvořen a otestován nástroj, který implementuje jak detekční, tak korekční algoritmy pro souborový systém UDF. Jeho funkce byla ověřena jak praktickým používáním na poškozená média, tak porovnáním výsledků detekce a následné korekce vůči ostatním existujícím řešením.

K vytvoření nástroje byl použit programovací jazyk C ve standardu C99 s použitím knihovních funkcí GNU/Linux. Jsou využívány výhradně funkce poskytované API GNU/Linux, což snižuje nároky na přeložení pro danou distribuci. Výčet požadovaných funkcí a knihoven je tento: `install`, `gawk`, `make`, `gcc` (nebo `clang`), `sed`, `grep`, `ld`, `nm`, `objdump`, `ar`, `ar`, `strip`, `ranlib`, `mt`, `ncurses`, `libreadline`, `libtool`, `automake`, `autoheader`, `autoconf`.

Nástroj v tuto chvíli implementuje podporu standardů UDF 1.00 až 2.01, přičemž většina vývoje byla soustředěna na novější verzi standardu. Proti tomu stojí nedostatek testovacích médií vůči kterým bylo zařízení testováno. Uměle vytvořená média (myšleno vytvořená pro účely testování) mají sice známou minulost ale to je částečně na škodu vzhledem k faktu, že opravdu závažné chyby vzniknou až během provozu média delší dobu.

Nástroj v tuto chvíli dokáže detekovat a opravovat chyby podle tabulky 6.1.

Chyba, ochranný mechanismus nebo funkce UDF, která není zachycena v této tabulce, není detekována.

Název chyby	Detekce	Korekce
Neshodující se kontrolní součet	✓	✓[1]
Neshodující se CRC	✓	✓[1]
Neshodující se umístění tagu	✓	✓
Neshodující se počet souborů nebo adresářů	✓[2]	✓[2]
Neshodující se objem volného místa	✓[2]	✓[2]
Neshodující se mapy obsazeného místa	✓[2]	✓[2]
Špatně nastavené *Unique ID*	✓	✓
Špatně nastavené časové značky modifikace	✓	✓
Špatně nastavené časové značky vytvoření	✗[4]	✗[4]
Rozdílné *Tag Serial Number*	✓	✓
Uzavření otevřeného deskriptoru LVID	✓	✓
Vyřešení nedokončeného zápisu	✓	✓
Poškozený deskriptor FE	✓	✗[3]
Poškozený deskriptor FID	✓	✗[3]
Poškozený deskriptor FSD	✓	✗[3]

Tab. 6.1: Tabulka detekovatelných a opravitelných chyb

[1] Pokud došlo k této poruše u redundantního deskriptoru

[2] Závislé na správnosti průchodu stromem souborů

[3] Oprava je sice technicky možná přepisem na správné hodnoty, ale není možné automaticky usoudit zda je při použití vadného deskriptoru v pořádku pokračovat. Tato možnost musí být svěřena pouze uživateli.

[4] Nebylo implementováno. Jedná se o porovnání pořadí času vytvoření a poslední modifikace.

7 PUBLIKACE NÁSTROJE V GNU KOMUNITĚ

Publikace v GNU komunitě proběhla díky začlenění vývojové větve zpět do zdrojového balíčku `udftools`. Předpoklady a kroky vedoucí k úspěšnému začlenění včetně dopadu po začlenění jsou popsány v následujících sekcích.

7.1 Integrace do balíčku `udftools`

Proces integrace do zdrojového balíčku je důležitou částí práce. Bez ní zůstane má práce pouze jako vývojová větev a nikdy se nedostane k uživatelům. Proto byla na začátku práce navázána komunikace s aktuálním správcem balíčku `udftools`, panem Palim Rohárem, který vlastní repozitář balíčku.

Vzhledem k nízké aktivitě ze strany vývojářů na balíčku `udftools` neexistuje seznam požadavků pro přispívání do balíčku, jako to je u jiných (aktivnějších) projektů. Tyto požadavky byly vyjednány s panem Rohárem a jednalo se o tyto:

- Funkční řešení přeložitelné pomocí GCC s využitím GNU Autotools.
- Kód okomentovaný a náležitě zdokumentovaný pro budoucí udržování.
- Vytvoření testů pro otestování funkčnosti v budoucnosti. Není potřeba vytvářet unit testy, ačkoli jsou vítány.
- Vytvoření manuálové stránky k programu.

Protože vývoj probíhá na serveru GitHub, samotná práce probíhala ve vlastní větvi, která vznikla rozdělením od původního projektu (*Fork*) a opačný proces, který větev začlení zpět, je nazýván *Pull Request*. To znamená, že vlastník repozitáře je vyzván ke sloučení vývojové větve s hlavní (*Merge*). To může proběhnout bez konfliktů, což je ideální stav, nebo dojde ke konfliktu se změnami v hlavní větvi. Pokud jsou konflikty vyřešeny a změny odpovídají požadavkům vlastníka repozitáře, změny jsou začleněny a vývojová větev je integrována.

7.2 Dopad integrace do balíčku `udftools`

Práce na `udffsck` byla od počátku koncipována jako součást existujícího balíčku `udftools`. Ten je začleněn ve většině distribucích GNU/Linux, potíž je, že často ve starých verzích, které jsou navázané na starý repozitář [8] místo stávajícího [9].

Pokud bychom tedy přepokládali integraci pouze do distribucí, které aktuálně používají verzi z nového repozitáře `udftools`, získali bychom tento seznam:

- **Debian** – aktuálně je balíček ve verzích Debian Testing a Debian Unstable. Více v [41].

- **Gentoo** – starší verze balíčku je v tuto chvíli Stable, poslední vydaná je Testing. Více v [38].
- **Arch Linux** – balíček je dodáván pomocí komunitního repozitáře AUR [37].
- **Ubuntu** – balíček je dodáván pomocí PPA. Je podporován ve verzích Ubuntu Zesty a Ubuntu Artful [40].
- **Fedora, Red Hat Enterprise** – poslední verze Fedora Linux obsahuje i poslední vydanou verzi balíčku, takže lze předpokládat její napojení na repozitář. Pravděpodobně bude tento balíček předáván i do RHEL, protože je známa aktivita v repozitáři ze strany společnosti Red Hat.
- **OpenSuse** – tato distribuce má v oficiálních repozitářích starou verzi balíčku, ale v [39] je k dispozici předkompilovaný balíček.

Jak je vidět, po začlenění se nástroj `udffsck` poměrně rychle dostane ke svým uživatelům skrz oficiální nebo komunitní distribuční kanály.

Dalšími kroky po úspěšné integraci bude průzkum pro další rozvoj nástroje. Prvními kroky pravděpodobně bude rozšíření podpory pro novější verze UDF. To bude pravděpodobně vyžadovat širší záběr než jen práci na `udffsck`, protože bude nutné přidat nové funkce z novějších verzí standardu UDF do sdílených hlavičkových souborů, což ovlivní i zbytek nástrojů v balíčku.

8 ZÁVĚR

Cílem práce bylo vytvořit nástroje pro diagnostiku integrity dat souborového systému pro OS Linux, pro který v současnosti tato podpora neexistuje.

Byla provedena analýza úložných řešení používaných v GNU/Linux z hlediska jejich struktury, funkce a možností kontroly konzistence. Bylo zjištěno, že nástroje pro kontrolu konzistence jsou zapouzdřeny pod nástrojem `fsck`, který volá specifické nástroje daného souborového systému. Cílem nástroje `fsck` je kontrola a oprava metadat (deskriptorů) souborového systému a ne oprava dat samotných. Oprava dat je možná pouze na některých souborových systémech, například ZFS používá nástroj `scrub`, který nahrazuje `fsck` a kontroluje nejen metadata ale i data souborového systému.

Během rešerše souborových systémů používaných v OS Linux bylo zjištěno, že neexistuje nástroj `fsck` pro souborový systém UDF. Proto jsem zvolil tento souborový systém pro další práci. Tomu předcházela analýza tohoto souborového systému z hlediska jeho funkce, použití a přístupu k detekci a korekci chyb.

Na základě výběru souborového systému UDF byly definovány detekovatelné a opravitelné poruchy, kterými má smysl se zabývat. V zásadě jde o chyby v metadatech souborového systému, protože data samotná jsou chráněna kontrolními a opravnými mechanismy nosného média, typicky pomocí ECC bloků. Metadata jsou na úrovni UDF chráněna kombinací kontroly skutečné a deklarované polohy na médiu, která umožňuje odhalit případné chyby vzniklé při zápisu deskriptoru, kontrolních součtů a CRC, jež zajišťují konzistenci deskriptorů jako takovou. Další implementovaný mechanismus souborového systému UDF je redundance deskriptorů. Tím je zvýšena šance na zachování nejdůležitějších deskriptorů souborového systému, tedy deskriptorů popisující médium jak celek. Klíčovou vlastností je vyřešení detekce a následné korekce neukončeného zápisu. Neukončený zápis lze detekovat v deskriptoru *Logical Volume Integrity Descriptor* nebo pomocí ostatních indikátorů jako například lišící se velikost souboru vůči jeho deklarované velikosti nebo nenastavné *Unique ID*. Všechny tyto způsoby jsou detailně popsány v kapitole 4. Poslední částí jsou korekce detekovaných chyb v deskriptorech stromu souborů. Tyto chyby úzce souvisí s přerušeným zápisem ale mohou vzniknout i jiným způsobem, například přehřátím buňky flash paměti nebo poškozeným kabelem vedoucím k médiu. Jedná se vedle chyb v deklarované poloze, kontrolním součtu a CRC o chyby v unikátním identifikátoru *Unique ID* každého souboru, respektive jeho deskriptorů *File Entry* a *File Identifier Descriptor*, chyby v sériovém čísle tagů deskriptorů, neaktualizovaný čas posledního zápisu nebo neshodující se deklarovaný a skutečný objem zaznamenaných dat.

Většinu detekovaných chyb je možné opravit. K tomu slouží korekční část ná-

stroje. Mezi opravitelné poruchy patří poškozený *Anchor Volume Descriptor Pointer*, což je vstupní bod do souborového systému. Stejně tak je opravitelná *Volume Descriptor Sequence*, která popisuje médium jak celek. Chyby provozního rázu v *Logical Volume Integrity Descriptor* opravit lze ale pokud je deskriptor poškozen strukturálně (t.j. selže porovnání kontrolního součtu nebo CRC), opravitelný v tuto chvíli není a bylo by nutné jej rekonstruovat. Opravy chyb ve stromu souborů se opět týkají nestrukturálních poruch. Pokud je deskriptor poškozen strkukturálně, je poškozená část souborového stromu odstraněna. Pokud se jedná o provozní poškození (například nekonzistence v sériových číslech tagů, neplatné časové značky nebo nekonzistentní *Unique ID*), je možné tyto chyby bez potíží odstranit.

Práce na nástroji `udffsck` byla započata v rámci existujícího balíčku `udftools`, který je součástí všech distribucí GNU/Linux. Je překvapivé, že do této doby nebyl tento nástroj vytvořen, vzhledem k absenci jiného funkčního korekčního řešení pro souborové systémy UDF. Nástroj `udffsck` jsem vytvořil v jazyce C podle standardu C99 s využitím knihovních funkcí poskytnutých operačním systémem GNU/Linux. Nástroj je přeložitelný s minimálními nároky na hostitelský systém stejně jako zbytek balíčku.

Mnou navržený nástroj `udffsck` slučuje jak detekční, tak korekční funkci. To znamená, že dokáže nejen detekovat všechny definované chyby na médiích naformátovaných souborovým systémem UDF až do verze standardu 2.01, ale také provádět jejich opravy. Implementace samotná intenzivně využívá mapování média do paměti a tím sdíleného paměťového prostoru (*Shared Memory*) pro zjednodušení práce s ním. Díky tomu byly eliminovány chyby vzniklé přesouváním ukazetele po médiu a je umožněno přímé mapování deskriptorů na médium.

Mnou vytvořené řešení v podobě nástroje `udffsck` bylo otestováno vůči výběrové množině testovacích dat o objemu 20 GB a 32 vzorcích, které zachycují nejčastěji se vyskytující verze UDF a jejich nejčastější nastavení. Většina testovacích dat byla vytvořena způsobem popsaným v podkapitole 6.1, t.j. vytvářením obrazů disků pomocí různých nástrojů poskytovaných prostředním GNU/Linux, Microsoft Windows a Apple macOS a jejich následným poškozováním a menší část byla převzata z testovací sady nástroje *blkid*. Významná část testovacích dat (přibližně o objemu 16 GB) byla vytvořena experimentálním používáním souborového systému s cílem napodobit životní cyklus média v reálném nasazení. Tento přístup pomohl odhalit řadu potíží, které nebylo možné odhalit syntetickým vytvářením poruch. Na základě těchto dat jsou vytvořeny automatizované testy, které jsou spouštěny po každém nahrání do repozitáře na server GitHub (viz. můj repozitář [10]) a testují, zda nedošlo k poškození dříve funkčních částí. Tímto způsobem bylo odhalena řada chyb, které by jinak mohly zůstat přehlédnuty. V rámci testování jsem otestoval širokou škálu možností, které souborový systém UDF poskytuje.

Výsledky práce byly publikovány v komunitě GNU v podobě začlenění výsledného nástroje `udffsck` do existujícího balíčku `udftools`. Tento proces předpokládal splnění nejen požadavků zadání této práce, ale i určité kvality kódu a jeho testů. Vzhledem k faktu, že od začátku tato práce směřovala k tomuto cíli, byla v rané fázi práce navázána komunikace se správcem repozitáře panem Palim Rohárem a postup práce byl konzultován s ním. Začleněním nástroje `udffsck` do balíčku `udftools` je zajištěno, že se nástroj dostane ke svým uživatelům v krátkém čase, vzhledem k napojení hlavních distribucí na tento repozitář.

Pokud mohu zhodnotit výsledky této práce, veřím že jsem splnil body zadání. Podařilo se vytvořit fungující nástroj pro kontrolu a korekci médií naformátovaných souborovým systémem UDF pro GNU/Linux, čímž se stává první takovou implementací s otevřeným zdrojovým kódem. Pokud mé řešení porovnám s komerčně dostupnými variantami, svou funkcí předčí například nástroj `CHKDSK` z prostředí Microsoft Windows, který nedokáže poškozené médium opravit tak, aby jej byl operační systém Microsoft Windows používat, zatímco mnou vytvořený nástroj toto dokáže. Nástroj pro kontrolu souborových systému v prostředí Apple macOS sice provádí opravy, ale neprovádí žádné korekce, tudíž nelze jeho činnost srovnat s činností mého nástroje.

LITERATURA

[1] *Universal Disk Format Specification*. Revision 2.60. Santa Clara, California: Optical Storage Technology Association, 2005.

[2] *Universal Disk Format Specification*. Revision 2.01. Cupertino, California: Optical Storage Technology Association, 2000.

[3] *Universal Disk Format Specification*. Revision 1.50. Santa Barbara, California: Optical Storage Technology Association, 1997.

[4] *ECMA-167*. 3rd Edition. Geneva, Switzerland: [online] ECMA, 1997 [cit. 2016-10-18]. Dostupné z: `https://www.ecma-international.org/publications/files/ECMA-ST/Ecma-167.pdf`

[5] *ECMA-168*. 2nd Edition. Geneva, Switzerland: [online] ECMA, 1994 [cit. 2017-05-07]. Dostupné z: `https://www.ecma-international.org/publications/files/ECMA-ST/Ecma-168.pdf`

[6] *Fsck tools for UDF. In: GitHub* [online]. 2016 [cit. 2016-09-30]. Dostupné z: `https://github.com/JElchison/format-udf/wiki/fsck-tools-for-UDF`

[7] *Internet Archive* [online]. 2007 [cit. 2016-11-21]. Dostupné z: `http://web.archive.org/web/20070702054203/http://www.hitech-projects.com/udf/download/`

[8] *Linux UDF. In: Sourceforge* [online]. 2004 [cit. 2016-11-21]. Dostupné z: `https://sourceforge.net/projects/linux-udf/`

[9] *pali/udftools. In: GitHub* [online]. 2016 [cit. 2016-11-21]. Dostupné z: `https://github.com/pali/udftools`

[10] *argorain/udftools. In: GitHub* [online]. 2017 [cit. 2017-04-21]. Dostupné z: `https://github.com/argorain/udftools`

[11] *UDF Filesystem* [online]. 2002 [cit. 2016-11-21]. Dostupné z: `https://people.freebsd.org/~scottl/`

[12] *13thmonkey.org: udfclient* [online]. 2016 [cit. 2016-11-21]. Dostupné z: `www.13thmonkey.org/udfclient`

[13] *illumos/illumos-gate. In: GitHub* [online]. 2016 [cit. 2016-11-21]. Dostupné z: `https://github.com/illumos/illumos-gate/tree/f7877f5d39900cfd8b20dd673e5ccc1ef7cc7447/usr/src/cmd/fs.d/udfs/fsck`

[14] Mmap. *The Single UNIX ® Specification, Version 2* [online]. The Open Group, 1997 [cit. 2017-01-02]. Dostupné z: http://pubs.opengroup.org/onlinepubs/7908799/xsh/mmap.html

[15] Software RAID and LVM. *ArchWiki* [online]. ArchLinux, 2016 [cit. 2017-01-02]. Dostupné z: https://wiki.archlinux.org/index.php/Software_RAID_and_LVM

[16] Filesystems. *The Linux Documentation Project* [online]. [cit. 2017-01-02]. Dostupné z: http://www.tldp.org/LDP/sag/html/filesystems.html

[17] The Second Extended File System. *Nongnu.org* [online]. 2011 [cit. 2017-01-02]. Dostupné z: http://www.nongnu.org/ext2-doc/ext2.html

[18] The ext filesystem – a four-generation retrospective. *Linux Magazine* [online]. Linux New Media, 2013 [cit. 2017-01-02]. Dostupné z: http://www.linux-magazine.com/Issues/2013/156/The-ext-Filesystem

[19] BADARI, Pulavarty, Mingming CAO, Theodore PULAVARTY a Suparna BHATTACHARYA. *State of the Art: Where we are with the Ext3 filesystem* [online]. 2005 [cit. 2017-01-02]. Dostupné z: http://ext2.sourceforge.net/2005-ols/paper-html/

[20] CAO, Mingming, Suparna BHATTACHARYA a Ted TSO. *Ext4: The Next Generation of Ext2/3 Filesystem* [online]. 2007, , 36 [cit. 2017-01-02]. Dostupné z: https://www.usenix.org/legacy/event/lsf07/tech/cao_m.pdf

[21] Fsck. *Die.net* [online]. [cit. 2017-01-02]. Dostupné z: https://linux.die.net/man/8/fsck

[22] Open. *Die.net* [online]. [cit. 2017-05-07]. Dostupné z: https://linux.die.net/man/2/open

[23] Fopen. *Die.net* [online]. [cit. 2017-05-07]. Dostupné z: https://linux.die.net/man/3/fopen

[24] Flock. *Die.net* [online]. [cit. 2017-05-07]. Dostupné z: https://linux.die.net/man/2/flock

[25] Mmap. *Die.net* [online]. [cit. 2017-05-07]. Dostupné z: https://linux.die.net/man/2/mmap

[26] Fstat. *Die.net* [online]. [cit. 2017-05-07]. Dostupné z: https://linux.die.net/man/2/fstat

[27] Fseeko. *Die.net* [online]. [cit. 2017-05-07]. Dostupné z: https://linux.die.net/man/3/fseeko

[28] Ftello. *Die.net* [online]. [cit. 2017-05-07]. Dostupné z: https://linux.die.net/man/3/ftello

[29] Fseek. *Die.net* [online]. [cit. 2017-05-07]. Dostupné z: https://linux.die.net/man/3/fseek

[30] Ftell. *Die.net* [online]. [cit. 2017-05-07]. Dostupné z: https://linux.die.net/man/3/ftell

[31] Read. *Die.net* [online]. [cit. 2017-05-07]. Dostupné z: https://linux.die.net/man/2/read

[32] Write. *Die.net* [online]. [cit. 2017-05-07]. Dostupné z: https://linux.die.net/man/2/write

[33] Isoinfo. *Die.net* [online]. [cit. 2017-01-02]. Dostupné z: https://linux.die.net/man/8/isoinfo

[34] NTFS for Mac 14. *Parargon Software Group* [online]. 2017 [cit. 2017-05-07]. Dostupné z: https://www.paragon-software.com/cz/home/ntfs-mac/

[35] UDF 2.60 - All approved DCNs. *Internet Archive* [online]. 2007 [cit. 2017-05-07]. Dostupné z: https://web.archive.org/web/20161119014819/http://www.osta.org/specs/pdf/udf260_allApprovedDCNs.pdf

[36] Datová komunikace. *UTKO* [online]. 2010 [cit. 2017-05-07]. Dostupné z: http://anca.utko.feec.vutbr.cz/vyuka/bdak

[37] Udftools. *Arch Linux User Repository* [online]. 2017 [cit. 2017-05-08]. Dostupné z: https://aur.archlinux.org/packages/udftools/

[38] sys-fs/udftools. *Gentoo Linux Packages* [online]. 2017 [cit. 2017-05-08]. Dostupné z: https://packages.gentoo.org/packages/sys-fs/udftools

[39] Udftools. *RPM Find* [online]. 2017 [cit. 2017-05-08]. Dostupné z: https://www.rpmfind.net/linux/rpm2html/search.php?query=udftools

[40] Udftools. *Launchpad* [online]. 2017 [cit. 2017-05-08]. Dostupné z: https://launchpad.net/ubuntu/+source/udftools/1.3-1

[41] Udftools. *Debian Package Tracker* [online]. 2017 [cit. 2017-05-08]. Dostupné z: https://tracker.debian.org/pkg/udftools

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

AD	Allocation Descriptor
AVDP	Anchor Volume Descriptor Pointer
CRC	Cyclic Redundancy Check
EFE	Extended File Entry
FE	File Entry
FID	File Identifier Descriptor
fsck	Filesystem Consistency Check
FSD	File Set Descriptor
ICB	Information Control Block
IUVD	Implementation Use Volume Descriptor
LVD	Logical Volume Descriptor
LVID	Logical Volume Integirty Descriptor
LVM	Logical Volume Manager
PVD	Primary Volume Descriptor
PD	Partition Descriptor
SBD	Space Bitmap Descriptor
TD	Terminating Descriptor
UDF	Universal Disk Filesystem
udffsck	UDF Filesystem Consistency Check
USD	Unallocated Space Descriptor
VDS	Volume Descriptor Sequence
VRS	Volume Recogniton Sequence

SEZNAM PŘÍLOH

A Výčet operačních systémů podporovaných v souborovém systému UDF **73**

B Manuálová stránka k `udffsck` **74**

C Ukázkový výpis programu `udffsck` **77**

D Výstupy z referenčního řešení `udfct` **80**

E Obsah přiloženého DVD **109**

A VÝČET OPERAČNÍCH SYSTÉMŮ PODPOROVANÝCH V SOUBOROVÉM SYSTÉMU UDF

Třída OS	Identifikátor OS	Operační systém
0	Libovolné	Nedefinváno
1	0	DOS/Windows 3.x
2	0	OS/2
3	0	Macintosh OS 9 a starší
3	1	Macintosh OS X a novější
4	0	obecný UNIX
4	1	UNIX – IBM AIX
4	2	UNIX – SUN OS/Solaris
4	3	UNIX – HP/UX
4	4	UNIX – Silicon Graphics Irix
4	5	UNIX – Linux
4	6	UNIX – MKLinux
4	7	UNIX – FreeBSD
4	8	UNIX – NetBSD
5	0	Windows 9x – obsahuje i Windows 98/ME
6	0	Windows NT – obsahuje Windows 2000, XP, Server 2003 a pozdější na stejné bázi
7	0	OS/400
8	0	BeOS
9	0	Windows CE

Tab. A.1: Seznam oficiálně podporovaných OS

B MANUÁLOVÁ STRÁNKA K UDFFSCK

NAME

udffsck - check and correction for UDF filesystem

SYNOPSIS

udffsck [-vvvciph] [-B BLOCKSIZE] medium

DESCRIPTION

udffsck is used to check and correct UDF file systems. There are known limitations regarding UDF version, which is now limited to 2.01 or older.

medium is the device file where file system is stored (e.g. /dev/sda1).

Please note it is not safe run udffsck on mounted file system. Even if you think it is safe to run it on mounted file system, whole report is invalid. You should never never make any changes on mounted file system, so if udffsck asks to do so, only valid answer is ,,no'', otherwise you can break your filesystem instead.

OPTIONS

-B BLOCKSIZE

Force udffsck to use this blocksize instead of autodetection. This value is in bytes. Default is autodetected value by finding VRS and AVDP positions.

-c Only check medium and print found errors. This is default behavior.

-i Interactively fix medium. In this mode all corrections must be authorized by user.

-p Automatical corrections. This is like -i, but all questions are answered yes.

-h Short help message.

-v Warning verbosity level. Errors and warning will be printed.

-vv Message verbosity level. Errors, warnings and messages will be printed. Recomended for manual usage.

-vvv Debug Verbosity level. Only for development and debug purposes. And for nosy users.

EXIT CODE

The exit code returned by udffsck is the sum of the following conditions:

0 - No errors
1 - File system errors corrected
2 - File system errors corrected, system should be rebooted (not used at this moment)
4 - File system errors left uncorrected
8 - Operational error
16 - Usage or syntax error
32 - udffsck canceled by user request
128 - Shared library error (not used at this moment)

EXAMPLES

Check medium and show it structure to user:

```
udffsck -vvc /dev/sda2
```

Check and fix medium image automatically, show only errors:

```
udffsck -p udf_image_file.img
```

BUGS

Reading and correcting is supported on UDF filesystems up to version 2.01. More recent filesystems are not currently covered and udffsck will end with exit code 8.

REPORTING BUGS

Almost any piece of software will have bugs. If you manage to find a filesystem which causes udffsck to crash, or which udffsck is unable to repair, please report it to the author.

Please include as much information as possible in bug report. It helps to analyze bug and hopefuly fix it.

Necessary information in bug report are udffsck version (obtainable by -h option) and OS information (uname -a). Also if possible attach compressed medium image which made udffsck fail and complete log from stdout and stderr at debug verbosity (obtainable by -vvv option).

AUTHOR

This version of udffsck was written by Vojtech Vladyka <vojtech.vladyka@gmail.com>

SEE ALSO

cdrwtool(1), fsck(8), mkudfs(8), pktsetup(8), wrudf(8)

C UKÁZKOVÝ VÝPIS PROGRAMU `UDFFSCK`

Výpis C.1: Ukázka výstupu programu pro médium s velikostí sektoru 2048 B a vstupními parametry `-vvp -B 2048`

```
Verbosity increased to WARNING.
Verbosity increased to MESSAGE.
Medium will be only checked. No corrections.
Device block size: 2048
Medium to analyze: ../../udf-samples/bs2048-r0201-clean.img
AVDP[0] successfully loaded.
AVDP[1] successfully loaded.
AVDP[2] successfully loaded.
Sectorsize: 2048

Stream file tree
----------------
.....:darwx.:.r.x.:.r.x STREAM  2017-04-12 09:50:16.201007+00:00        40  <ROOT>

Medium file tree
----------------
.....:..rwx.:.r.x.:.r.x DIR     2017-04-12 07:50:16.201007+00:00       188  <ROOT>
  .d...:..rwx.:.r.x.:.r.x DIR     2017-04-12 07:50:16.201007+00:00        40  "lost
      +found"
  .d...:..rwx.:.r.x.:.r.x DIR     2017-04-12 07:52:57.795796+00:00        96  "data
      "
    .d...:..rwx.:.r.x.:.r.x DIR     2017-04-12 14:14:58.964404+00:00       416  "
        life␣is␣strange"
      .....:..rw..:.r...:.r.. FILE    2017-04-12 07:53:37.304305+00:00    572192  "
          life␣is␣strange␣by␣katewindhelm.jpg"
      .....:..rw..:.r...:.r.. FILE    2017-04-12 07:53:37.304305+00:00    352154  "
          642030.jpg"
      ..D..           <Unused FID>                                          "life␣
          is␣strange.jpg"
      .....:..rw..:.r...:.r.. FILE    2017-04-12 07:53:37.344305+00:00   1588217  "
          Life␣is␣strange.jpg"
      .....:..rw..:.r...:.r.. FILE    2017-04-12 07:53:37.360306+00:00    899265  "
          Chloe␣background.jpg"
  .d...:..rwx.:.r.x.:.r.x DIR     2017-04-12 07:53:12.051979+00:00        96  "
      nejaka␣data"
    .....:..rw..:.r...:.r.. FILE    2017-04-12 07:53:51.380488+00:00    153600  "
        acminecraft.tar"

Filesystem status
-----------------
Volume set identifier:58eddc38LinuxUDF
Partition identifier:LinuxUDF
Next UniqueID: 26
Max found UniqueID: 25
Last LVID recoreded change: 2017-04-12 14:15:11.380535+00:00
expected number of files: 5
expected number of dirs:  5
counted number of files: 5
counted number of dirs:  5
UDF rev: min read:  0201
         min write: 0201
         max write: 0201
```

```
43 Used Space: 3596288 (1756)
44 Free Space: 15796224 (7713)
45 Partition size: 19392512 (9469)
46 Expected Used Space: 3596288 (1756)
47 Expected Used Blocks: 1756
48 Expected Unused Blocks: 7713
49
50 VDS verification status
51 -----------------------
52 [0] PVD is fine. No fixing needed.
53 [1] LVD is fine. No fixing needed.
54 [2] PD is fine. No fixing needed.
55 [3] USD is fine. No fixing needed.
56 [4] IUVD is fine. No fixing needed.
57 [5] TD is fine. No fixing needed.
58 All done
```

Legenda (čísla řádků)

1. a 2. Výpis nastavení výpisů
3. Oznámení módu chodu programu
4. Přijatá velikost bloku vstupním argumentem
5. Analyzované médium
6. – 8. Oznámení o úspěšnosti načtení *Anchor Volume Descriptor Pointer*
9. Detekovaná velikost bloku
11. – 13. Výpis stromu souborů *Stream Directory*
15. – 27. Výpis stromu souborů. Každý řádek odpovídá jednomu záznamu (soubor nebo adresář). Prvních 5 příznaků jsou *File Characteristics*, ty jsou použity například na řádku 18, kde písmeno `d` značí adresář nebo na řádku 23, kde písmeno `D` značí smazaný soubor. Poté následují opravnění práce se záznamem v pořadí smazatelnost, editovatelnost atributů, čtení, zápis spouštění. Ty jsou ve skupinách práv pro uživatele, skupinu a ostatní. Poté následuje kategorie záznamu (soubor, adresář, stream). Další je datum poslední změny následovaný velikostí souboru v bytech. Jako poslední je identifikátor záznamu.
29. Výpis kontroly souborového systému
31. Identifikátor média
32. Identifikátor logického oddílu
33. Příští *Unique ID*, které bude použito pro další záznam
34. Nejvyšší nalezené *Unique ID* na médiu
35. Poslední zaznamenaná změna v *Logical Volume Integrity Descriptor*
36. – 39. Očekáváné a nalezené počty souborů a adresářů
40. – 42. Povolené revize UDF pro práci s médiem
43. – 48. Detekované a deklarované objemy volného a obsazeného místa v bytech a blocích
50. – 57. Výsledek kontroly a opravy *Volume Descriptor Sequence*

Takto program postupně zpracuje všechny soubory až skončí s hláškou *All done*. Pokud vše skončilo správně, měla by být jeho návratová hodnota 0, což je i tento případ. Pokud by došlo k nějaké opravě, bude k návratové hodnotě přičtena hodnota 1, pokud byly nalezeny chyby a nebyly opraveny, bude přičteno 4. Pokud by program zhavaroval, bude návratová hodnota 8.

Více výstupů z běhu programu je v příloze na CD, adresář `udffsck-outputs`.

D VÝSTUPY Z REFERENČNÍHO ŘEŠENÍ `udfct`

Výpis D.1: Výsledek kontroly poškozeného média referenčním programem `udfct`

```
UDF Conformance Testing Application
(c) Koninklijke Philips Electronics N.V. 1999-2006
Application version : 1.5r4 distribution bin
   UCT Core version : 1.5r4 distribution bin
       platform : Linux - with scsi support

Command:
  ./udf_test
Generic options parsing:

Medium info options parsing:
    -blocksize 512

Device options parsing:
    ../../../udffsck-test-samples/udf-samples/
        bs512_windows7_udf0201_broken_file_tree.img

No image configuration file ../../../udffsck-test-samples/udf-samples/
    bs512_windows7_udf0201_broken_file_tree.cfg


Created image device
  image file blocks : 3913728  ../../../udffsck-test-samples/udf-samples/
      bs512_windows7_udf0201_broken_file_tree.img
     blocks, bytes  : 3913728  2003828736


'-image' device Medium Info:
  last valid block  : 3913727   Volume Space: 1.8662 Gbyte
  block size        : 512
  (ECC) blocking factor : 0
  nmb of sessions   : 1
  verify session    : 1
  session starts    : 0
  medium WR type    : unknown
  medium SE type    : unknown
  medium CL type    : unknown finalization state

=> Warning: Undefined (ECC) blocking factor, set to 16.
-       Please specify "-ecclength␣<n>" to enable the verifier to do
-       a better job. Media that do not have ECC or fixed size packets
-       must specify: -ecclength 1

Inspect 1 block for presence of VAT or AVDP
starting at block: 3913727

  ==>   read cache: max 32 buffers of 32 sectors, total 512 Kb
3913727         1 ok block read
3913727 AVDP    (MVDS: 96, RVDS: 3913696)

last AVDP at 3913727 (N)

    Single Layer medium

```

```
Verification start medium info
  last valid block  : 3913727   Volume Space: 1.8662 Gbyte
  block size        : 512
  (ECC) blocking factor : 16
  nmb of sessions   : 1
  verify session    : 1
  session starts    : 0
  medium WR type    : unknown
  medium SE type    : unknown
  medium CL type    : unknown finalization state

====>   Start verification
    Start time   : 2017-05-09 10:41:32 +02:00 (east of UTC)
    Verbose level: 100
    Message limit: 20
    Fake read enabled
    Read cache enabled
    Initial UDF Revision range: 1.02 thru 2.60
    Single Layer medium

====>   Volume Structure verification
    Read Volume Recognition Sequence
     64 read 4 blocks
    BEA01
    Start of Extended Area
     68 read 4 blocks
    NSR03
  ==>   Changed UDF Revision range from: 1.02 thru 2.60 to: 2.00 thru 2.60
-   because of "NSR03␣descriptor"
     72 read 4 blocks
    TEA01
     76 read 4 blocks
    End of Extended Area
    End of Volume Recognition Sequence


    Reading Volume Information
    256 read block
    AVDP at 256 (MVDS: 96, RVDS: 3913696)
    First Tag Serial Number: 2
3913471 read block
    AVDP at N-256   (MVDS: 96, RVDS: 3913696)
3913727 read block
    AVDP at N   (MVDS: 96, RVDS: 3913696)

    Number of AVDPs: 3, AVDPs at  256,  N-256,  N

====>   Read Main VDS extent:      96, length:   8192
     96 read block
    PVD   VDS Number: 1
    PVD   Recording Time: 2017-05-05 17:58:30.010400 +02:00
    PVD   Volume Identifier    : "UDF␣Volume"
    PVD   Volume Set Identifier: "113A1E01␣UDF␣Volume␣Set"
  ==>   PVD  344 New Entity Identifier (regid):
    Application Entity Identifier
      <empty>
  ==>   PVD  388 New Entity Identifier (regid):
    Implementation Entity Identifier
      Identifier         : "*Microsoft␣Windows"
```

```
      OS Class       : #06  Windows NT
      OS Identifier      : #00  Windows NT - generic
  ==>   Add PVD to VDS info
     97 read block
    PD    VDS Number: 2, Partition Number: 8192
  ==>   PD   24  New Entity Identifier (regid):
    Application Entity Identifier
      Identifier         : "+NSR03"
    Unallocated Space Bitmap
  ==>   Add PD to VDS info, partition number: 8192
     98 read block
    LVD   VDS Number: 3
    LVD   Logical Volume Identifier: "windows7_udf_r2.01"
  ==>   LVD  216 New Entity Identifier (regid):
    Domain Entity Identifier
      Identifier         : "*OSTA␣UDF␣Compliant"
      UDF revision       : 2.01
  ==>   Changed UDF Revision range from: 2.00 thru 2.60 to: 2.01 only
-   because of "Domain␣EntityID␣UDF␣revision"
      Domain flags       : #00
    LVD   FSD at: (0,p0)
  ==>   Add LVD to VDS info
     99 read block
    USD   VDS Number: 4, nmb of ADs: 7
     1: location      76 thru        95,       20 blocks
     2: location     112 thru       127,       16 blocks
     3: location     144 thru       255,      112 blocks
     4: location     257 thru       287,       31 blocks
     5: location 3913440 thru   3913470,       31 blocks
     6: location 3913472 thru   3913695,      224 blocks
     7: location 3913712 thru   3913726,       15 blocks
  ==>   Add USD to VDS info
    100 read block
    IUVD  VDS Number: 5
    IUVD EntityID Identifier: "*UDF␣LV␣Info"
  ==>   IUVD 20  New Entity Identifier (regid):
    UDF Entity Identifier
      Identifier         : "*UDF␣LV␣Info"
      UDF revision       : 2.01
      OS Class       : #06  Windows NT
      OS Identifier      : #00  Windows NT - generic
    UDF IUVD Logical Volume Identifier : "windows7_udf_r2.01"
    UDF IUVD LVInfo1: <undefined>
    UDF IUVD LVInfo2: <undefined>
    UDF IUVD LVInfo3: <undefined>
  ==>   Add IUVD to VDS info
    101 read block
    TD

====>   Read Reserve VDS extent: 3913696, length:   8192
3913696 read block
    PVD   VDS Number: 1
    PVD   Recording Time: 2017-05-05 17:58:30.010400 +02:00
    PVD   Volume Identifier    : "UDF␣Volume"
    PVD   Volume Set Identifier: "113A1E01␣UDF␣Volume␣Set"
  ==>   Add PVD to VDS info
3913697 read block
    PD    VDS Number: 2, Partition Number: 8192
    Unallocated Space Bitmap
```

```
  ==>   Add PD to VDS info, partition number: 8192
3913698 read block
    LVD   VDS Number: 3
    LVD   Logical Volume Identifier: "windows7_udf_r2.01"
    LVD   FSD at: (0,p0)
  ==>   Add LVD to VDS info
3913699 read block
    USD   VDS Number: 4, nmb of ADs: 7
     1: location      76 thru        95,       20 blocks
     2: location     112 thru       127,       16 blocks
     3: location     144 thru       255,      112 blocks
     4: location     257 thru       287,       31 blocks
     5: location 3913440 thru   3913470,       31 blocks
     6: location 3913472 thru   3913695,      224 blocks
     7: location 3913712 thru   3913726,       15 blocks
  ==>   Add USD to VDS info
3913700 read block
    IUVD  VDS Number: 5
    IUVD EntityID Identifier: "*UDF␣LV␣Info"
    UDF IUVD Logical Volume Identifier : "windows7_udf_r2.01"
    UDF IUVD LVInfo1: <undefined>
    UDF IUVD LVInfo2: <undefined>
    UDF IUVD LVInfo3: <undefined>
  ==>   Add IUVD to VDS info
3913701 read block
    TD

====>   Check equivalence of Main VDS and Reserve VDS

  ==>   Main and Reserve VDS are equivalent

====>   Check Main VDS. Summary:
     PVD VDS Number  1
     LVD VDS Number  3
     USD VDS Number  4
      PD VDS Number  2
    IUVD VDS Number  5  ID: "*UDF␣LV␣Info"
       5 prevailing VDS descriptors found


  ==>   Using Main VDS

  ==>   Checking VDS 'far␣apart' allocation for Main and Reserve VDS.

   =>   Volume Descriptor Sequence 'far␣apart' test summary:
-         Main VDS LBA range:         96  thru        111,   1 extent
-      Reserve VDS LBA range:    3913696  thru    3913711,   1 extent
-     lowest packet distance:  244600 ECC packets (100.0% of remaining Volume Space
    )
-        test margin level 1:   24500 ECC packets ( 10.0% of remaining Volume Space
    )
-        test margin level 2:   61200 ECC packets ( 25.0% of remaining Volume Space
    )
-          Main VDS    Reserve VDS    ECC packet distance
-              111        3913696       244600


====>   Checking Logical Volume: "windows7_udf_r2.01"

```

```
226     Prevailing Partition Descriptors:
227       pNmb: 8192, start:   288, length: 3913152, access: overwritable
228
229     LVD Partition Maps:
230       p0: Physical Partition Map (Type 1), pNmb: 8192
231
232   ==>   Changed medium WR type from unknown to overwritable
233 -           because of partition access type
234   ==>   Changed medium SE type from unknown to nonsequential
235 -           because of no Virtual Partition found
236
237
238     Mounted Partitions:
239 -   p0: Physical, pNmb: 8192, blocks:     288 thru 3913439, access: overwritable
240 -             logical blocks:       0 thru 3913151
241
242   ==>   Changed medium CL type from unknown finalization state to finalized
243 -           because of more than one AVDP found
244
245     Read LVID sequence extent: 128, length: 8192
246     128 read block
247     LVID  - Open
248     LVID 8 Error: Descriptor CRC: #24B2, expected: #03CC, ECMA 3/7.2.6, 4/7.2.6
249     LVID Recording Time: 2017-05-05 16:15:01.278032
250   ==>   LVID 88  New Entity Identifier (regid):
251     Implementation Entity Identifier
252       Identifier         : "*Microsoft␣Windows␣␣␣␣␣"
253       OS Class           : #04  UNIX
254       OS Identifier      : #05  UNIX - Linux
255       Implementation Use : #00 #00 #06 #01 #01 #00
256     129 read block
257     TD
258     LVID 28 Error: Dirty Volume. An Open prevailing LVID was
259 -         found but no Virtual Partition. The Logical Volume
260 -         is in an inconsistent state, unexpected errors
261 -         may occur, ECMA 3/8.8.2, 3/10.10.3.
262 -   Note: The verifier will assume the volume to have a Close LVID
263 -         in order to show all LVID and volume inconsistencies.
264
265     Next UniqueID: #0000000000000012,
266     from LVID Logical Volume Header Descriptor.
267
268
269   ==>   p0: read Unallocated or Freed Partition Space Sets
270         Unallocated Space Bitmap extent: 956 blocks
271     320 read block
272     321 read 955 blocks
273     SBD
274     SBD  10     Note: Descriptor CRC Length: 8. This is the recommended value
275 -             according to errata DCN-5108 for UDF 2.50 and lower.
276     p0: Space set found
277     --
278
279     Read FSD sequence extent: (0,p0), length:    512
280
281     288 read block
282     FSD   FSN: 0,  FSDN: 0
283     FSD   Logical Volume Identifier: "windows7_udf_r2.01"
```

```
284     FSD          File Set Identifier: "UDF␣Volume␣Set"
285     FSD    Copyright File Identifier: <undefined>
286     FSD     Abstract File Identifier: <undefined>
287     FSD            Root Directory at: (988,p0)
288     FSD  System Stream Directory at: <undefined>
289
290 ====>   Volume identifiers summary:
291
292         PVD:          Volume Identifier  [32]: "UDF␣Volume"
293         PVD:     Volume Set Identifier [128]: "113A1E01␣UDF␣Volume␣Set"
294         LVD: Logical Volume Identifier [128]: "windows7_udf_r2.01"
295        IUVD: Logical Volume Identifier [128]: "windows7_udf_r2.01"
296         FSD: Logical Volume Identifier [128]: "windows7_udf_r2.01"
297         FSD:       File Set Identifier  [32]: "UDF␣Volume␣Set"
298
299 ====>   File Structure verification
300
301     Read Root Directory
302    1276 read block
303     EFE  file type DIR    name: <root>
304     EFE 80 Error: FE Access Time shall not be later than
305 -         LVID Recording Time, ECMA 3/10.10.2, UDF 2.2.6, UDF 3.1.1.
306 -              FE Access Time: 2017-05-05 16:15:01.282032
307 -         LVID Recording Time: 2017-05-05 16:15:01.278032
308 -         name: <root>
309   ==>   EFE  168 New Entity Identifier (regid):
310     Implementation Entity Identifier
311       Identifier          : "*Linux␣UDFFS"
312       OS Class       : #04  UNIX
313       OS Identifier       : #05  UNIX - Linux
314     EFE  216 Embedded data, 152 bytes
315
316   ==>   (max) depth:  1  1  Expand directory: <root>
317
318     EFE  216 Verify embedded FIDs, 152 bytes
319     FID   cid:       name: /<parent FID>, refers to: <root>
320     FID   cid:   8   name: "gtd.mp4"
321     FID   cid:   8   name: "2017-01-16-191632.webm"
322     Error: Tag Serial Number changing: 2 -> 1, no disaster
323 -           recovery support, ECMA 3/7.2.5, 4/7.2.5, UDF 2.3.1.1.
324 -         name: "2017-01-16-191632.webm"
325     Add FIDs to directory hierarchy and read FEs
326     289 read block
327     EFE  file type FILE   name: "gtd.mp4"
328     Error: Tag Serial Number changing: 1 -> 2, no disaster
329 -           recovery support, ECMA 3/7.2.5, 4/7.2.5, UDF 2.3.1.1.
330 -         name: "gtd.mp4"
331 - ==>   Message printed 2 times, ignored from now.
332     EFE 80 Error: FE Access Time shall not be later than
333 -         LVID Recording Time, ECMA 3/10.10.2, UDF 2.2.6, UDF 3.1.1.
334 -              FE Access Time: 2017-05-05 16:15:06.790101
335 -         LVID Recording Time: 2017-05-05 16:15:01.278032
336 -         name: "gtd.mp4"
337     cnt:  extent type,      size,  location,part,     body, total alloc
338       1: short_ad   0  327345425      39131 0    327345425  327345664
339    1277 read block
340     EFE  file type FILE   name: "2017-01-16-191632.webm"
```

```
    EFE 80 Error: FE Access Time shall not be later than
-         LVID Recording Time, ECMA 3/10.10.2, UDF 2.2.6, UDF 3.1.1.
-              FE Access Time: 2017-05-05 16:15:31.094407
-         LVID Recording Time: 2017-05-05 16:15:01.278032
-         name: "2017-01-16-191632.webm"
    EFE 92 Error: FE Modification Time shall not be later than
-         LVID Recording Time, ECMA 3/10.10.2, UDF 2.2.6, UDF 3.1.1.
-         FE Modification Time: 2017-05-05 16:15:31.330410
-           LVID Recording Time: 2017-05-05 16:15:01.278032
-         name: "2017-01-16-191632.webm"
    EFE 116 Error: FE Attribute Time shall not be later than
-         LVID Recording Time, ECMA 3/10.10.2, UDF 2.2.6, UDF 3.1.1.
-            FE Attribute Time: 2017-05-05 16:15:31.330410
-         LVID Recording Time: 2017-05-05 16:15:01.278032
-         name: "2017-01-16-191632.webm"
    EFE 104 Error: EFE Creation Time shall not be later than
-         LVID Recording Time, ECMA 3/10.10.2, UDF 2.2.6, UDF 3.1.1.
-            EFE Creation Time: 2017-05-05 16:15:31.094407
-         LVID Recording Time: 2017-05-05 16:15:01.278032
-         name: "2017-01-16-191632.webm"
    cnt:  extent type,      size,  location,part,     body, total alloc
      1: long_ad    0    9830400         990 0       9830400    9830400
      2: long_ad    1       1024       20190 0       9830400    9831424
    EFE 72 Error: Logical Blocks Recorded: 19202, expected: 19200,
-             icbtag Allocation Descriptor type: 1,
-             UDF 2.3.6.5, ECMA 4/14.9.11, 4/14.6.8.
-         name: "2017-01-16-191632.webm"

Directory: <root>

.d.p.e.........:..rwx:..rwx:..rwx DIR   1 2017-05-05 15:58        152 /<parent FID
    >, refers to: <root>
.....S.........:..rwx:..rwx:..rwx FILE  1 2009-11-12 19:46  327345425 "gtd.mp4"
.....L.........:..rw.:..r..:..r.. FILE  1 2017-05-05 16:15    9830400 "
    2017-01-16-191632.webm"

    file body read: "gtd.mp4"
  39419 fake read 639347 blocks
    file body read: "2017-01-16-191632.webm"
   1278 fake read 19200 blocks
    file tail read: "2017-01-16-191632.webm"
  20478 fake read 2 blocks

    Expand complete, max depth  1 for directory: <root>
-             2 files   0 directories
-    overall total:    2 files   1 directory

    Maximum directory depth: 1

    End of directory tree expansion
    Excluding deleted FIDs with cleared ICB

====>   Testing File Link Count by cross reference of 4 paths.
    File Link Count errors will be identified here by the
    physical address of the File Entry as well as all
    paths identifying the File Entry.
    The physical address of the File Entry is also shown in
```

```
396     the informational read block messages above.
397     Note that errors found here may have been reported before
398     or may be caused by other previously reported errors.
399
400 ====>   Testing free Volume Space in USD Allocation Descriptors
401
402 ====>   Build Partition Space Bitmaps.
403     Also check structures that overlap with partition space.
404
405 ====>   Partition Allocation summary :
406
407  ===>   Physical Partition p0:  size 3913152 blocks, overwritable
408                 blocks     288 thru 3913439
409
410   ==>   Compare partition p0 calculated bitmap to recorded Space Set
411       using: Unallocated Space Bitmap
412
413       All used blocks marked as allocated
414
415       Warning: 222981 unused blocks NOT marked as unallocated.
416 -           This may be due to previous errors,
417 -           or otherwise it is Orphan Space, UDF 5.2.2.
418 -     Usage of following extents not yet identified:
419   20480   logical block    20192 thru    39130,   18939 blocks
420  678766   logical block   678478 thru   882519,  204042 blocks
421
422 ====>   Final LVID verification
423     Dirty LVID, verify as Close
424   ==>   overwritable Physical Partition p0 Space summary:
425                         Partition Length    : 3913152
426                   LVID Partition Size       : 3913152
427                   LVID Partition Free Space: 3253643
428        Unallocated Space Bitmap free space: 3030662
429                Verifier expected free space: 3253643
430         Unused by UDF but allocated blocks:  222981
431
432     LVID 80 Error: LVID and Unallocated Space Bitmap inconsistent
433 -             Partition Free Space for a Dirty LVID,
434 -             see Space summary above and maybe use
435 -             -showalloc output, ECMA 3/10.1, UDF 2.2.6.2.
436
437   ==>           LVID status summary:
438     Last modification Time     : 2017-05-05 16:15:01.278032 (equal to UTC)
439     Last written Developer Id : "*Microsoft␣Windows␣␣␣␣␣"
440     Next UniqueID              : #0000000000000012 => from dirty LVID
441     max used FE  UniqueID      : #0000000000000011
442     max used FID UniqueID      :          #00000011
443     Number of Files            :         2
444     Number of Directories      :         1
445     Min UDF Read   Revision    : UDF 2.01
446     Min UDF Write  Revision    : UDF 2.01
447     Max UDF Write  Revision    : UDF 2.01
448         Medium UDF Revision    : UDF 2.01
449
450
451 ====>   Testing uniqueness of relevant UniqueIDs.
452
453
```

```
    Test complete
    Elapsed time : 00:00

====>   Volume identifiers summary:

        PVD:         Volume Identifier  [32]: "UDF␣Volume"
        PVD:     Volume Set Identifier [128]: "113A1E01␣UDF␣Volume␣Set"
        LVD: Logical Volume Identifier [128]: "windows7_udf_r2.01"
       IUVD: Logical Volume Identifier [128]: "windows7_udf_r2.01"
        FSD: Logical Volume Identifier [128]: "windows7_udf_r2.01"
        FSD:       File Set Identifier  [32]: "UDF␣Volume␣Set"

====>   Encountered EntityID (regid) summary:

 count  EntityID

    3   Domain Entity Identifier
      Identifier         : "*OSTA␣UDF␣Compliant"
      UDF revision       : 2.01
      Domain flags       : #00
    2   UDF Entity Identifier
      Identifier         : "*UDF␣LV␣Info"
      UDF revision       : 2.01
      OS Class       : #06  Windows NT
      OS Identifier      : #00  Windows NT - generic
    3   Implementation Entity Identifier
      Identifier         : "*Linux␣UDFFS"
      OS Class       : #04  UNIX
      OS Identifier      : #05  UNIX - Linux
    8   Implementation Entity Identifier
      Identifier         : "*Microsoft␣Windows"
      OS Class       : #06  Windows NT
      OS Identifier      : #00  Windows NT - generic
    1   Implementation Entity Identifier
      Identifier         : "*Microsoft␣Windows␣␣␣␣␣"
      OS Class       : #04  UNIX
      OS Identifier      : #05  UNIX - Linux
      Implementation Use : #00 #00 #06 #01 #01 #00
    2   Application Entity Identifier
      <empty>
    2   Application Entity Identifier
      Identifier         : "+NSR03"

  These EntityIDs are also shown above when read for the first time

====>   Final verify status report

  Final UDF Revision range: 2.01 only
  Disaster Recovery for Tag Serial Numbers not supported

  File System info
  last valid block  : 3913727   Volume Space: 1.8662 Gbyte
  block size        : 512
  (ECC) blocking factor : 16
  nmb of sessions   : 1
  verify session    : 1
  session starts    : 0
  medium WR type    : overwritable
  medium SE type    : nonsequential
```

```
513   medium CL type    : finalized
514
515   Summed file body sizes: 337175825 bytes   (321.5559 Mbytes)
516
517     Error count:   6    total occurrences:    12  -> search for   "error:"
518   Warning count:   2    total occurrences:     2  -> search for "warning:"
519
520   Additional notes may have been printed:     -> search for    "note:"
521
522   Disclaimer:
523 -   The number of errors and warnings is an indication only.
524 -   There is no guarantee that the number of errors and
525 -   warnings as shown by the UDF verifier is correct.
```

Výpis D.2: Výsledek kontroly poškozeného média referenčním programem `udfct` po opravě nástrojem `CHKDSK`

```
UDF Conformance Testing Application
(c) Koninklijke Philips Electronics N.V. 1999-2006
Application version : 1.5r4 distribution bin
   UCT Core version : 1.5r4 distribution bin
       platform : Linux - with scsi support

Command:
  ./udf_test
Generic options parsing:

Medium info options parsing:
    -blocksize 512

Device options parsing:
    ../../../bs512_windows7_udf0201_chkdsk.img

No image configuration file ../../../bs512_windows7_udf0201_chkdsk.cfg


Created image device
  image file blocks : 3913728  ../../../bs512_windows7_udf0201_chkdsk.img
     blocks, bytes  : 3913728  2003828736


'-image' device Medium Info:
  last valid block  : 3913727   Volume Space: 1.8662 Gbyte
  block size        : 512
  (ECC) blocking factor : 0
  nmb of sessions   : 1
  verify session    : 1
  session starts    : 0
  medium WR type    : unknown
  medium SE type    : unknown
  medium CL type    : unknown finalization state

=> Warning: Undefined (ECC) blocking factor, set to 16.
-       Please specify "-ecclength␣<n>" to enable the verifier to do
-       a better job. Media that do not have ECC or fixed size packets
-       must specify: -ecclength 1

Inspect 1 block for presence of VAT or AVDP
starting at block: 3913727

  ==>   read cache: max 32 buffers of 32 sectors, total 512 Kb
3913727         1 ok block read
3913727 AVDP    (MVDS: 96, RVDS: 3913721)

last AVDP at 3913727 (N)

    Single Layer medium

Verification start medium info
  last valid block  : 3913727   Volume Space: 1.8662 Gbyte
  block size        : 512
  (ECC) blocking factor : 16
  nmb of sessions   : 1
```

```
  verify session    : 1
  session starts    : 0
  medium WR type    : unknown
  medium SE type    : unknown
  medium CL type    : unknown finalization state

====>   Start verification
    Start time   : 2017-05-09 09:52:52 +02:00 (east of UTC)
    Verbose level: 100
    Message limit: 20
    Fake read enabled
    Read cache enabled
    Initial UDF Revision range: 1.02 thru 2.60
    Single Layer medium

====>   Volume Structure verification
    Read Volume Recognition Sequence
     64 read 4 blocks
    BEA01
    Start of Extended Area
     68 read 4 blocks
    NSR03
  ==>   Changed UDF Revision range from: 1.02 thru 2.60 to: 2.00 thru 2.60
-   because of "NSR03␣descriptor"
     72 read 4 blocks
    TEA01
     76 read 4 blocks
    End of Extended Area
    End of Volume Recognition Sequence


    Reading Volume Information
    256 read block
    AVDP at 256 (MVDS: 96, RVDS: 3913721)
    First Tag Serial Number: 2
    AVDP error: Main or Reserve Volume Descriptor Sequence Extent
-           has extent length less than 16 logical blocks (8192 bytes)
-           Main: 8192 bytes, Reserve: 3072 bytes, UDF 2.2.3.1+2
3913471 read block
    AVDP at N-256   (MVDS: 96, RVDS: 3913721)
    AVDP error: Main or Reserve Volume Descriptor Sequence Extent
-           has extent length less than 16 logical blocks (8192 bytes)
-           Main: 8192 bytes, Reserve: 3072 bytes, UDF 2.2.3.1+2
3913727 read block
    AVDP at N   (MVDS: 96, RVDS: 3913721)
    AVDP error: Main or Reserve Volume Descriptor Sequence Extent
-           has extent length less than 16 logical blocks (8192 bytes)
-           Main: 8192 bytes, Reserve: 3072 bytes, UDF 2.2.3.1+2

    Number of AVDPs: 3, AVDPs at  256,  N-256,  N

====>   Read Main VDS extent:      96, length:   8192
     96 read block
    PVD   VDS Number: 1
    PVD   Recording Time: 2017-05-05 17:58:30.010400 +02:00
    PVD   Volume Identifier    : "UDF␣Volume"
    PVD   Volume Set Identifier: "113A1E01␣UDF␣Volume␣Set"
```

```
  ==>   PVD  344 New Entity Identifier (regid):
    Application Entity Identifier
      <empty>
  ==>   PVD  388 New Entity Identifier (regid):
    Implementation Entity Identifier
      Identifier         : "*Microsoft␣Windows"
      OS Class       : #06  Windows NT
      OS Identifier      : #00  Windows NT - generic
  ==>   Add PVD to VDS info
     97 read block
    PD    VDS Number: 2, Partition Number: 8192
  ==>   PD   24  New Entity Identifier (regid):
    Application Entity Identifier
      Identifier         : "+NSR03"
    Unallocated Space Bitmap
  ==>   Add PD to VDS info, partition number: 8192
     98 read block
    LVD   VDS Number: 3
    LVD   Logical Volume Identifier: "windows7_udf_r2.01"
  ==>   LVD  216 New Entity Identifier (regid):
    Domain Entity Identifier
      Identifier         : "*OSTA␣UDF␣Compliant"
      UDF revision       : 2.01
  ==>   Changed UDF Revision range from: 2.00 thru 2.60 to: 2.01 only
-   because of "Domain␣EntityID␣UDF␣revision"
      Domain flags       : #00
    LVD   FSD at: (0,p0)
  ==>   Add LVD to VDS info
     99 read block
    USD   VDS Number: 4, nmb of ADs: 6
     1: location      76 thru        95,       20 blocks
     2: location     112 thru       159,       48 blocks
     3: location     176 thru       255,       80 blocks
     4: location     257 thru       287,       31 blocks
     5: location 3913440 thru   3913470,       31 blocks
     6: location 3913472 thru   3913720,      249 blocks
  ==>   Add USD to VDS info
    100 read block
    IUVD  VDS Number: 5
    IUVD EntityID Identifier: "*UDF␣LV␣Info"
  ==>   IUVD 20  New Entity Identifier (regid):
    UDF Entity Identifier
      Identifier         : "*UDF␣LV␣Info"
      UDF revision       : 2.01
      OS Class       : #06  Windows NT
      OS Identifier      : #00  Windows NT - generic
    UDF IUVD Logical Volume Identifier : "windows7_udf_r2.01"
    UDF IUVD LVInfo1: <undefined>
    UDF IUVD LVInfo2: <undefined>
    UDF IUVD LVInfo3: <undefined>
  ==>   Add IUVD to VDS info
    101 read block
    TD

====>   Read Reserve VDS extent: 3913721, length:   3072
3913721 read block
    PVD   VDS Number: 1
    PVD   Recording Time: 2017-05-05 17:58:30.010400 +02:00
    PVD   Volume Identifier    : "UDF␣Volume"
```

```
    PVD   Volume Set Identifier: "113A1E01␣UDF␣Volume␣Set"
  ==>   Add PVD to VDS info
3913722 read block
    PD    VDS Number: 2, Partition Number: 8192
    Unallocated Space Bitmap
  ==>   Add PD to VDS info, partition number: 8192
3913723 read block
    LVD   VDS Number: 3
    LVD   Logical Volume Identifier: "windows7_udf_r2.01"
    LVD   FSD at: (0,p0)
  ==>   Add LVD to VDS info
3913724 read block
    USD   VDS Number: 4, nmb of ADs: 6
     1: location      76 thru        95,       20 blocks
     2: location     112 thru       159,       48 blocks
     3: location     176 thru       255,       80 blocks
     4: location     257 thru       287,       31 blocks
     5: location 3913440 thru   3913470,       31 blocks
     6: location 3913472 thru   3913720,      249 blocks
  ==>   Add USD to VDS info
3913725 read block
    IUVD  VDS Number: 5
    IUVD EntityID Identifier: "*UDF␣LV␣Info"
    UDF IUVD Logical Volume Identifier : "windows7_udf_r2.01"
    UDF IUVD LVInfo1: <undefined>
    UDF IUVD LVInfo2: <undefined>
    UDF IUVD LVInfo3: <undefined>
  ==>   Add IUVD to VDS info
3913726 read block
    TD

====>   Check equivalence of Main VDS and Reserve VDS

  ==>   Main and Reserve VDS are equivalent

====>   Check Main VDS. Summary:
     PVD VDS Number  1
     LVD VDS Number  3
     USD VDS Number  4
      PD VDS Number  2
    IUVD VDS Number  5  ID: "*UDF␣LV␣Info"
       5 prevailing VDS descriptors found


  ==>   Using Main VDS

  ==>   Checking VDS 'far␣apart' allocation for Main and Reserve VDS.

   =>   Volume Descriptor Sequence 'far␣apart' test summary:
-         Main VDS LBA range:         96  thru        111,   1 extent
-      Reserve VDS LBA range:    3913721  thru    3913726,   1 extent
-     lowest packet distance:  244601 ECC packets (100.0% of remaining Volume Space
    )
-        test margin level 1:   24500 ECC packets ( 10.0% of remaining Volume Space
    )
-        test margin level 2:   61200 ECC packets ( 25.0% of remaining Volume Space
    )
-          Main VDS    Reserve VDS    ECC packet distance
-              111        3913721       244601
```

```
229
230
231 ====>   Checking Logical Volume: "windows7_udf_r2.01"
232
233     Prevailing Partition Descriptors:
234       pNmb: 8192, start:   288, length: 3913152, access: overwritable
235
236     LVD Partition Maps:
237       p0: Physical Partition Map (Type 1), pNmb: 8192
238
239   ==>   Changed medium WR type from unknown to overwritable
240 -           because of partition access type
241   ==>   Changed medium SE type from unknown to nonsequential
242 -           because of no Virtual Partition found
243
244
245     Mounted Partitions:
246 -   p0: Physical, pNmb: 8192, blocks:     288 thru 3913439, access: overwritable
247 -             logical blocks:       0 thru 3913151
248
249   ==>   Changed medium CL type from unknown finalization state to finalized
250 -           because of more than one AVDP found
251
252     Read LVID sequence extent: 160, length: 8192
253     160 read block
254     LVID  - Close
255     LVID Recording Time: 2017-05-08 18:01:57 +02:00
256     161 read block
257     TD
258
259     Next UniqueID: #0000000000000001,
260     from LVID Logical Volume Header Descriptor.
261
262     Error: Next UniqueID lower 32 bits value less than 16, UDF 3.2.1.1.
263
264   ==>   p0: read Unallocated or Freed Partition Space Sets
265         Unallocated Space Bitmap extent: 956 blocks
266     320 read block
267     321 read 955 blocks
268     SBD
269     SBD  10     Note: Descriptor CRC Length: 8. This is the recommended value
270 -             according to errata DCN-5108 for UDF 2.50 and lower.
271     p0: Space set found
272     --
273
274     Read FSD sequence extent: (0,p0), length:    512
275
276     288 read block
277     FSD   FSN: 0,  FSDN: 0
278     FSD   Logical Volume Identifier: "windows7_udf_r2.01"
279     FSD          File Set Identifier: "UDF␣Volume␣Set"
280     FSD   Copyright File Identifier: <undefined>
281     FSD    Abstract File Identifier: <undefined>
282     FSD            Root Directory at: (988,p0)
283     FSD  System Stream Directory at: <undefined>
284
285 ====>   Volume identifiers summary:
286
```

```
         PVD:          Volume Identifier  [32]: "UDF␣Volume"
         PVD:     Volume Set Identifier [128]: "113A1E01␣UDF␣Volume␣Set"
         LVD: Logical Volume Identifier [128]: "windows7_udf_r2.01"
        IUVD: Logical Volume Identifier [128]: "windows7_udf_r2.01"
         FSD: Logical Volume Identifier [128]: "windows7_udf_r2.01"
         FSD:       File Set Identifier  [32]: "UDF␣Volume␣Set"

====>   File Structure verification

    Read Root Directory
   1276 read block
    EFE  file type DIR    name: <root>
    EFE 8 Error: Descriptor CRC: #4958, expected: #7AD0, ECMA 3/7.2.6, 4/7.2.6
-        name: <root>
  ==>   EFE  168 New Entity Identifier (regid):
    Implementation Entity Identifier
      Identifier         : "*Linux␣UDFFS"
      OS Class       : #04  UNIX
      OS Identifier      : #05  UNIX - Linux
    EFE  216 Embedded data, 152 bytes

  ==>   (max) depth:  1  1  Expand directory: <root>

    EFE  216 Verify embedded FIDs, 152 bytes
    FID   cid:       name: /<parent FID>, refers to: <root>
    FID   cid:   8   name: "gtd.mp4"
    FID 32 Error: UniqueID: #00000010
-          Next UniqueID: #0000000000000001
-        FID UniqueID shall be less than Next UniqueID,
-        UDF 3.2.1.1, ECMA 4/14.15.1.
-        name: "gtd.mp4"
    FID   cid:   8   name: "2017-01-16-191632.webm"
    FID 32 Error: UniqueID: #00000011
-          Next UniqueID: #0000000000000001
-        FID UniqueID shall be less than Next UniqueID,
-        UDF 3.2.1.1, ECMA 4/14.15.1.
-        name: "2017-01-16-191632.webm"
    Add FIDs to directory hierarchy and read FEs
    289 read block
    EFE  file type FILE   name: "gtd.mp4"
    EFE 200 Error: UniqueID: #0000000000000010
-          Next UniqueID: #0000000000000001
-        EFE UniqueID shall be less than Next UniqueID,
-        UDF 3.2.1.1, ECMA 4/14.15.1.
-        name: "gtd.mp4"
    cnt:  extent type,      size,  location,part,     body, total alloc
      1: short_ad   0  327345425      39131 0    327345425  327345664
   1277 read block
    EFE  file type FILE   name: "2017-01-16-191632.webm"
    EFE 200 Error: UniqueID: #0000000000000011
-          Next UniqueID: #0000000000000001
-        EFE UniqueID shall be less than Next UniqueID,
-        UDF 3.2.1.1, ECMA 4/14.15.1.
-        name: "2017-01-16-191632.webm"
    cnt:  extent type,      size,  location,part,     body, total alloc
      1: long_ad    0    9830400        990 0      9830400    9830400
      2: long_ad    1       1024      20190 0      9830400    9831424
```

```
344
345 Directory: <root>
346
347 .d.p.e.....:..rwx:..rwx:..rwx DIR   1 2017-05-05 15:58        152 /<parent FID
        >, refers to: <root>
348 .....S.....:..rwx:..rwx:..rwx FILE  1 2009-11-12 19:46  327345425 "gtd.mp4"
349 .....L.....:..rw.:..r..:..r.. FILE  1 2017-05-05 16:15    9830400 "
        2017-01-16-191632.webm"
350
351     file body read: "gtd.mp4"
352   39419 fake read 639347 blocks
353     file body read: "2017-01-16-191632.webm"
354    1278 fake read 19200 blocks
355     file tail read: "2017-01-16-191632.webm"
356   20478 fake read 2 blocks
357
358     Expand complete, max depth  1 for directory: <root>
359 -              2 files   0 directories
360 -    overall total:    2 files   1 directory
361
362     Maximum directory depth: 1
363
364     End of directory tree expansion
365     Excluding deleted FIDs with cleared ICB
366
367 ====>   Testing File Link Count by cross reference of 4 paths.
368     File Link Count errors will be identified here by the
369     physical address of the File Entry as well as all
370     paths identifying the File Entry.
371     The physical address of the File Entry is also shown in
372     the informational read block messages above.
373     Note that errors found here may have been reported before
374     or may be caused by other previously reported errors.
375
376 ====>   Testing free Volume Space in USD Allocation Descriptors
377
378 ====>   Build Partition Space Bitmaps.
379     Also check structures that overlap with partition space.
380
381 ====>   Partition Allocation summary :
382
383  ===>   Physical Partition p0:  size 3913152 blocks, overwritable
384                 blocks     288 thru 3913439
385
386   ==>   Compare partition p0 calculated bitmap to recorded Space Set
387       using: Unallocated Space Bitmap
388
389       Error: 658551 used blocks marked as unallocated or freed
390     289   logical block        1
391    1277   logical block      989 thru    20191,   19203 blocks
392   39419   logical block    39131 thru   678477,  639347 blocks
393
394       No unused blocks marked as allocated.
395
396 ====>   Final LVID verification
397     Close LVID
398   ==>   overwritable Physical Partition p0 Space summary:
399                       Partition Length    : 3913152
400                   LVID Partition Size     : 3913152
```

```
                  LVID Partition Free Space: 3912194
        Unallocated Space Bitmap free space: 3912194
               Verifier expected free space: 3253643
         Used by UDF but unallocated blocks:  658551

    LVID 80 FreeSpaceTable error: Physical Partition p0 Free Space: 3912194,
-         expected: 3253643, see Space summary above and
-         maybe use -showalloc output, UDF 2.2.6.2, ECMA 3/10.10.
-   Note: The verifier may be unable to find the correct values for
-         some of these LVID/VAT tests because of previous errors.

  ==>           LVID status summary:
    Last modification Time    : 2017-05-08 18:01:57 +02:00 (east of UTC)
    Last written Developer Id : "*Microsoft␣Windows"
    Next UniqueID             : #0000000000000001 => from LVID
    max used FE  UniqueID     : #0000000000000011
    max used FID UniqueID     :          #00000011
    Number of Files           :         0
    Number of Directories     :         0
    Min UDF Read   Revision   : UDF 2.01
    Min UDF Write  Revision   : UDF 2.01
    Max UDF Write  Revision   : UDF 2.01
        Medium UDF Revision   : UDF 2.01

    LVID 120 Error: Number of Files: 0, expected: 2, UDF 2.2.6.4.
    LVID 124 Error: Number of Directories: 0, expected: 1, UDF 2.2.6.4.

====>   Testing uniqueness of relevant UniqueIDs.


    Test complete
    Elapsed time : 00:00

====>   Volume identifiers summary:

        PVD:          Volume Identifier  [32]: "UDF␣Volume"
        PVD:     Volume Set Identifier [128]: "113A1E01␣UDF␣Volume␣Set"
        LVD: Logical Volume Identifier [128]: "windows7_udf_r2.01"
       IUVD: Logical Volume Identifier [128]: "windows7_udf_r2.01"
        FSD: Logical Volume Identifier [128]: "windows7_udf_r2.01"
        FSD:       File Set Identifier  [32]: "UDF␣Volume␣Set"

====>   Encountered EntityID (regid) summary:

 count  EntityID

    3   Domain Entity Identifier
      Identifier         : "*OSTA␣UDF␣Compliant"
      UDF revision       : 2.01
      Domain flags       : #00
    2   UDF Entity Identifier
      Identifier         : "*UDF␣LV␣Info"
      UDF revision       : 2.01
      OS Class       : #06  Windows NT
      OS Identifier      : #00  Windows NT - generic
    3   Implementation Entity Identifier
      Identifier         : "*Linux␣UDFFS"
      OS Class       : #04  UNIX
```

```
459       OS Identifier      : #05  UNIX - Linux
460     9   Implementation Entity Identifier
461       Identifier         : "*Microsoft␣Windows"
462       OS Class       : #06  Windows NT
463       OS Identifier      : #00  Windows NT - generic
464     2   Application Entity Identifier
465       <empty>
466     2   Application Entity Identifier
467       Identifier         : "+NSR03"
468
469   These EntityIDs are also shown above when read for the first time
470
471 ====>   Final verify status report
472
473   Final UDF Revision range: 2.01 only
474
475   File System info
476   last valid block  : 3913727   Volume Space: 1.8662 Gbyte
477   block size        : 512
478   (ECC) blocking factor : 16
479   nmb of sessions   : 1
480   verify session    : 1
481   session starts    : 0
482   medium WR type    : overwritable
483   medium SE type    : nonsequential
484   medium CL type    : finalized
485
486   Summed file body sizes: 337175825 bytes   (321.5559 Mbytes)
487
488     Error count:   8    total occurrences:    13  -> search for   "error:"
489   Warning count:   1    total occurrences:     1  -> search for "warning:"
490
491   Additional notes may have been printed:     -> search for    "note:"
492
493   Disclaimer:
494 -   The number of errors and warnings is an indication only.
495 -   There is no guarantee that the number of errors and
496 -   warnings as shown by the UDF verifier is correct.
```

Výpis D.3: Výsledek kontroly poškozeného média referenčním programem `udfct` po pokusu o jeho opravu po `CHKDSK` mým nástrojem `udffsck`

```
UDF Conformance Testing Application
(c) Koninklijke Philips Electronics N.V. 1999-2006
Application version : 1.5r4 distribution bin
   UCT Core version : 1.5r4 distribution bin
       platform : Linux - with scsi support

Command:
  ./udf_test
Generic options parsing:

Medium info options parsing:
    -blocksize 512

Device options parsing:
    /dev/sdb

No image configuration file /dev/sdb.cfg


Created image device
  image file blocks : 3913728  /dev/sdb
     blocks, bytes  : 3913728  2003828736


'-image' device Medium Info:
  last valid block  : 3913727   Volume Space: 1.8662 Gbyte
  block size        : 512
  (ECC) blocking factor : 0
  nmb of sessions   : 1
  verify session    : 1
  session starts    : 0
  medium WR type    : unknown
  medium SE type    : unknown
  medium CL type    : unknown finalization state

=> Warning: Undefined (ECC) blocking factor, set to 16.
-       Please specify "-ecclength <n>" to enable the verifier to do
-       a better job. Media that do not have ECC or fixed size packets
-       must specify: -ecclength 1

Inspect 1 block for presence of VAT or AVDP
starting at block: 3913727

  ==>   read cache: max 32 buffers of 32 sectors, total 512 Kb
3913727         1 ok block read
3913727 AVDP    (MVDS: 96, RVDS: 3913721)

last AVDP at 3913727 (N)

    Single Layer medium

Verification start medium info
  last valid block  : 3913727   Volume Space: 1.8662 Gbyte
  block size        : 512
  (ECC) blocking factor : 16
  nmb of sessions   : 1
```

```
  verify session    : 1
  session starts    : 0
  medium WR type    : unknown
  medium SE type    : unknown
  medium CL type    : unknown finalization state

====>   Start verification
    Start time   : 2017-05-09 13:19:12 +02:00 (east of UTC)
    Verbose level: 100
    Message limit: 20
    Fake read enabled
    Read cache enabled
    Initial UDF Revision range: 1.02 thru 2.60
    Single Layer medium

====>   Volume Structure verification
    Read Volume Recognition Sequence
     64 read 4 blocks
    BEA01
    Start of Extended Area
     68 read 4 blocks
    NSR03
  ==>   Changed UDF Revision range from: 1.02 thru 2.60 to: 2.00 thru 2.60
-   because of "NSR03␣descriptor"
     72 read 4 blocks
    TEA01
     76 read 4 blocks
    End of Extended Area
    End of Volume Recognition Sequence


    Reading Volume Information
    256 read block
    AVDP at 256 (MVDS: 96, RVDS: 3913721)
    First Tag Serial Number: 2
3913471 read block
    AVDP at N-256   (MVDS: 96, RVDS: 3913721)
3913727 read block
    AVDP at N   (MVDS: 96, RVDS: 3913721)

    Number of AVDPs: 3, AVDPs at  256,  N-256,  N

====>   Read Main VDS extent:      96, length:   8192
     96 read block
    PVD   VDS Number: 1
    PVD   Recording Time: 2017-05-05 17:58:30.010400 +02:00
    PVD   Volume Identifier    : "UDF␣Volume"
    PVD   Volume Set Identifier: "113A1E01␣UDF␣Volume␣Set"
  ==>   PVD  344 New Entity Identifier (regid):
    Application Entity Identifier
      <empty>
  ==>   PVD  388 New Entity Identifier (regid):
    Implementation Entity Identifier
      Identifier         : "*Microsoft␣Windows"
      OS Class       : #06  Windows NT
      OS Identifier      : #00  Windows NT - generic
  ==>   Add PVD to VDS info
     97 read block
    PD    VDS Number: 2, Partition Number: 8192
```

```
  ==>   PD   24  New Entity Identifier (regid):
    Application Entity Identifier
      Identifier         : "+NSR03"
    Unallocated Space Bitmap
  ==>   Add PD to VDS info, partition number: 8192
     98 read block
    LVD   VDS Number: 3
    LVD   Logical Volume Identifier: "windows7_udf_r2.01"
  ==>   LVD  216 New Entity Identifier (regid):
    Domain Entity Identifier
      Identifier         : "*OSTA␣UDF␣Compliant"
      UDF revision       : 2.01
  ==>   Changed UDF Revision range from: 2.00 thru 2.60 to: 2.01 only
-   because of "Domain␣EntityID␣UDF␣revision"
      Domain flags       : #00
    LVD   FSD at: (0,p0)
  ==>   Add LVD to VDS info
     99 read block
    USD   VDS Number: 4, nmb of ADs: 6
     1: location      76 thru        95,       20 blocks
     2: location     112 thru       159,       48 blocks
     3: location     176 thru       255,       80 blocks
     4: location     257 thru       287,       31 blocks
     5: location 3913440 thru   3913470,       31 blocks
     6: location 3913472 thru   3913720,      249 blocks
  ==>   Add USD to VDS info
    100 read block
    IUVD  VDS Number: 5
    IUVD EntityID Identifier: "*UDF␣LV␣Info"
  ==>   IUVD 20  New Entity Identifier (regid):
    UDF Entity Identifier
      Identifier         : "*UDF␣LV␣Info"
      UDF revision       : 2.01
      OS Class       : #06  Windows NT
      OS Identifier      : #00  Windows NT - generic
    UDF IUVD Logical Volume Identifier : "windows7_udf_r2.01"
    UDF IUVD LVInfo1: <undefined>
    UDF IUVD LVInfo2: <undefined>
    UDF IUVD LVInfo3: <undefined>
  ==>   Add IUVD to VDS info
    101 read block
    TD

====>   Read Reserve VDS extent: 3913721, length:   8192
    Error: Last sector 3913736 of VDS extent beyond Volume Space, ECMA 3/8.5.
3913721 read block
    PVD   VDS Number: 1
    PVD   Recording Time: 2017-05-05 17:58:30.010400 +02:00
    PVD   Volume Identifier    : "UDF␣Volume"
    PVD   Volume Set Identifier: "113A1E01␣UDF␣Volume␣Set"
  ==>   Add PVD to VDS info
3913722 read block
    PD    VDS Number: 2, Partition Number: 8192
    Unallocated Space Bitmap
  ==>   Add PD to VDS info, partition number: 8192
3913723 read block
    LVD   VDS Number: 3
    LVD   Logical Volume Identifier: "windows7_udf_r2.01"
```

```
174     LVD   FSD at: (0,p0)
175   ==>   Add LVD to VDS info
176 3913724 read block
177     USD   VDS Number: 4, nmb of ADs: 6
178      1: location      76 thru        95,       20 blocks
179      2: location     112 thru       159,       48 blocks
180      3: location     176 thru       255,       80 blocks
181      4: location     257 thru       287,       31 blocks
182      5: location 3913440 thru   3913470,       31 blocks
183      6: location 3913472 thru   3913720,      249 blocks
184   ==>   Add USD to VDS info
185 3913725 read block
186     IUVD  VDS Number: 5
187     IUVD EntityID Identifier: "*UDF␣LV␣Info"
188     UDF IUVD Logical Volume Identifier : "windows7_udf_r2.01"
189     UDF IUVD LVInfo1: <undefined>
190     UDF IUVD LVInfo2: <undefined>
191     UDF IUVD LVInfo3: <undefined>
192   ==>   Add IUVD to VDS info
193 3913726 read block
194     TD
195
196 ====>   Check equivalence of Main VDS and Reserve VDS
197
198   ==>   Main and Reserve VDS are equivalent
199
200 ====>   Check Main VDS. Summary:
201      PVD VDS Number  1
202      LVD VDS Number  3
203      USD VDS Number  4
204       PD VDS Number  2
205     IUVD VDS Number  5  ID: "*UDF␣LV␣Info"
206        5 prevailing VDS descriptors found
207
208
209   ==>   Using Main VDS
210
211   ==>   Checking VDS 'far␣apart' allocation for Main and Reserve VDS.
212
213    =>   Volume Descriptor Sequence 'far␣apart' test summary:
214 -         Main VDS LBA range:         96  thru        111,   1 extent
215 -       Reserve VDS LBA range:    3913721  thru    3913736,   1 extent
216 -      lowest packet distance:  244601 ECC packets (100.0% of remaining Volume Space
    )
217 -         test margin level 1:   24500 ECC packets ( 10.0% of remaining Volume Space
    )
218 -         test margin level 2:   61200 ECC packets ( 25.0% of remaining Volume Space
    )
219 -           Main VDS    Reserve VDS    ECC packet distance
220 -               111         3913721        244601
221
222
223 ====>   Checking Logical Volume: "windows7_udf_r2.01"
224
225     Prevailing Partition Descriptors:
226       pNmb: 8192, start:   288, length: 3913152, access: overwritable
227
228     LVD Partition Maps:
229       p0: Physical Partition Map (Type 1), pNmb: 8192
```

```
230
231   ==>   Changed medium WR type from unknown to overwritable
232 -           because of partition access type
233   ==>   Changed medium SE type from unknown to nonsequential
234 -           because of no Virtual Partition found
235
236
237     Mounted Partitions:
238 -   p0: Physical, pNmb: 8192, blocks:     288 thru 3913439, access: overwritable
239 -             logical blocks:       0 thru 3913151
240
241   ==>   Changed medium CL type from unknown finalization state to finalized
242 -           because of more than one AVDP found
243
244     Read LVID sequence extent: 160, length: 8192
245     160 read block
246     LVID  - Close
247     LVID Recording Time: 2017-05-09 12:32:10.768729 +02:00
248   ==>   LVID 88  New Entity Identifier (regid):
249     Implementation Entity Identifier
250       Identifier         : "*Microsoft␣Windows␣␣␣␣␣"
251       OS Class       : #06  Windows NT
252       OS Identifier      : #00  Windows NT - generic
253       Implementation Use : #00 #00 #06 #01 #01 #00
254     161 read block
255     TD
256
257     Next UniqueID: #0000000000000013,
258     from LVID Logical Volume Header Descriptor.
259
260
261   ==>   p0: read Unallocated or Freed Partition Space Sets
262         Unallocated Space Bitmap extent: 956 blocks
263     320 read block
264     321 read 955 blocks
265     SBD
266     SBD  10     Note: Descriptor CRC Length: 8. This is the recommended value
267 -             according to errata DCN-5108 for UDF 2.50 and lower.
268     p0: Space set found
269     --
270
271     Read FSD sequence extent: (0,p0), length:    512
272
273     288 read block
274     FSD   FSN: 0,  FSDN: 0
275     FSD   Logical Volume Identifier: "windows7_udf_r2.01"
276     FSD         File Set Identifier: "UDF␣Volume␣Set"
277     FSD   Copyright File Identifier: <undefined>
278     FSD    Abstract File Identifier: <undefined>
279     FSD           Root Directory at: (988,p0)
280     FSD  System Stream Directory at: <undefined>
281
282 ====>   Volume identifiers summary:
283
284         PVD:         Volume Identifier  [32]: "UDF␣Volume"
285         PVD:     Volume Set Identifier [128]: "113A1E01␣UDF␣Volume␣Set"
286         LVD: Logical Volume Identifier [128]: "windows7_udf_r2.01"
287        IUVD: Logical Volume Identifier [128]: "windows7_udf_r2.01"
288         FSD: Logical Volume Identifier [128]: "windows7_udf_r2.01"
```

```
        FSD:       File Set Identifier  [32]: "UDF␣Volume␣Set"

====>   File Structure verification

    Read Root Directory
   1276 read block
    EFE  file type DIR    name: <root>
    EFE  216 Embedded data, 260 bytes

  ==>   (max) depth:  1  1  Expand directory: <root>

    EFE  216 Verify embedded FIDs, 260 bytes
    FID   cid:       name: /<parent FID>, refers to: <root>
    FID   cid:   8   name: "gtd.mp4"
    FID   cid:   8   name: "2017-01-16-191632.webm"
    FID   cid: 255 D name: "Nov#00E1␣slo#017Eka"
    Note: deleted FID, compression ID 255
- ==>   Message printed once, ignored from now.
    FID   cid:   8   name: "asd"
    Add FIDs to directory hierarchy and read FEs
    289 read block
    EFE  file type FILE   name: "gtd.mp4"
  ==>   EFE  168 New Entity Identifier (regid):
    Implementation Entity Identifier
      Identifier         : "*Linux␣UDFFS"
      OS Class       : #04  UNIX
      OS Identifier      : #05  UNIX - Linux
    cnt:  extent type,      size,  location,part,     body, total alloc
      1: short_ad   0  327345425      39131 0    327345425  327345664
   1277 read block
    EFE  file type FILE   name: "2017-01-16-191632.webm"
    cnt:  extent type,      size,  location,part,     body, total alloc
      1: long_ad    0    9830400        990 0      9830400    9830400
      2: long_ad    1       1024      20190 0      9830400    9831424
    290 read block
    EFE  file type DIR    name: "asd"
    EFE  216 Embedded data, 40 bytes

Directory: <root>

.d.p.e.........:..rwx:..rwx:..rwx DIR   2 2017-05-09 10:32        260 /<parent FID
    >, refers to: <root>
.....S.........:..rwx:..rwx:..rwx FILE  1 2009-11-12 19:46  327345425 "gtd.mp4"
.....L.........:..rw.:..r..:..r.. FILE  1 2017-05-05 16:15    9830400 "
    2017-01-16-191632.webm"
.dD............      <unused FID>                                      "Nov#00E1␣
    slo#017Eka"
.d...e.........:darwx:darwx:darwx DIR   1 2017-05-09 10:32         40 "asd"

    file body read: "gtd.mp4"
  39419 fake read 639347 blocks
    file body read: "2017-01-16-191632.webm"
   1278 fake read 19200 blocks
    file tail read: "2017-01-16-191632.webm"
  20478 fake read 2 blocks

  ==>   (max) depth:  2  2  Expand directory: "asd"

    EFE  216 Verify embedded FIDs, 40 bytes
```

```
345     FID   cid:       name: "asd"/<parent FID>, refers to: <root>
346     Add FIDs to directory hierarchy and read FEs
347
348 Directory: /"asd"
349
350 .d.p.e.........:..rwx:..rwx:..rwx DIR   2 2017-05-09 10:32        260 "asd"/<
        parent FID>, refers to: <root>
351
352
353     Expand complete, max depth  2 for directory: "asd"
354 -              0 files   0 directories
355
356     Expand complete, max depth  2 for directory: <root>
357 -              2 files   1 directory
358 -    overall total:    2 files   2 directories
359
360     Maximum directory depth: 2
361
362     End of directory tree expansion
363     Excluding deleted FIDs with cleared ICB
364
365 ====>   Testing File Link Count by cross reference of 6 paths.
366     File Link Count errors will be identified here by the
367     physical address of the File Entry as well as all
368     paths identifying the File Entry.
369     The physical address of the File Entry is also shown in
370     the informational read block messages above.
371     Note that errors found here may have been reported before
372     or may be caused by other previously reported errors.
373
374 ====>   Testing free Volume Space in USD Allocation Descriptors
375
376 ====>   Build Partition Space Bitmaps.
377     Also check structures that overlap with partition space.
378
379 ====>   Partition Allocation summary :
380
381  ===>   Physical Partition p0:  size 3913152 blocks, overwritable
382                 blocks     288 thru 3913439
383
384   ==>   Compare partition p0 calculated bitmap to recorded Space Set
385       using: Unallocated Space Bitmap
386
387       Error: 2 used blocks marked as unallocated or freed
388   20478   logical block    20190 thru    20191,       2 blocks
389
390       No unused blocks marked as allocated.
391
392 ====>   Final LVID verification
393     Close LVID
394   ==>   overwritable Physical Partition p0 Space summary:
395                       Partition Length    : 3913152
396                  LVID Partition Size      : 3913152
397                  LVID Partition Free Space: 3253644
398        Unallocated Space Bitmap free space: 3253644
399               Verifier expected free space: 3253642
400         Used by UDF but unallocated blocks:       2
401
402     LVID 80 FreeSpaceTable error: Physical Partition p0 Free Space: 3253644,
```

```
-         expected: 3253642, see Space summary above and
-         maybe use -showalloc output, UDF 2.2.6.2, ECMA 3/10.10.
-   Note: The verifier may be unable to find the correct values for
-         some of these LVID/VAT tests because of previous errors.

  ==>          LVID status summary:
    Last modification Time    : 2017-05-09 12:32:10.768729 +02:00 (east of UTC)
    Last written Developer Id : "*Microsoft␣Windows␣␣␣␣␣"
    Next UniqueID             : #0000000000000013 => from LVID
    max used FE  UniqueID     : #0000000000000012
    max used FID UniqueID     :          #00000012
    Number of Files           :         2
    Number of Directories     :         2
    Min UDF Read   Revision   : UDF 2.01
    Min UDF Write  Revision   : UDF 2.01
    Max UDF Write  Revision   : UDF 2.01
        Medium UDF Revision   : UDF 2.01


====>   Testing uniqueness of relevant UniqueIDs.


    Test complete
    Elapsed time : 00:00

====>   Volume identifiers summary:

        PVD:          Volume Identifier  [32]: "UDF␣Volume"
        PVD:     Volume Set Identifier [128]: "113A1E01␣UDF␣Volume␣Set"
        LVD: Logical Volume Identifier [128]: "windows7_udf_r2.01"
       IUVD: Logical Volume Identifier [128]: "windows7_udf_r2.01"
        FSD: Logical Volume Identifier [128]: "windows7_udf_r2.01"
        FSD:       File Set Identifier  [32]: "UDF␣Volume␣Set"

====>   Encountered EntityID (regid) summary:

 count  EntityID

    3   Domain Entity Identifier
      Identifier         : "*OSTA␣UDF␣Compliant"
      UDF revision       : 2.01
      Domain flags       : #00
    2   UDF Entity Identifier
      Identifier         : "*UDF␣LV␣Info"
      UDF revision       : 2.01
      OS Class       : #06  Windows NT
      OS Identifier      : #00  Windows NT - generic
    2   Implementation Entity Identifier
      Identifier         : "*Linux␣UDFFS"
      OS Class       : #04  UNIX
      OS Identifier      : #05  UNIX - Linux
    8   Implementation Entity Identifier
      Identifier         : "*Microsoft␣Windows"
      OS Class       : #06  Windows NT
      OS Identifier      : #00  Windows NT - generic
    3   Implementation Entity Identifier
      Identifier         : "*Microsoft␣Windows␣␣␣␣␣"
      OS Class       : #06  Windows NT
```

```
      OS Identifier      : #00  Windows NT - generic
      Implementation Use : #00 #00 #06 #01 #01 #00
    2   Application Entity Identifier
      <empty>
    2   Application Entity Identifier
      Identifier         : "+NSR03"

  These EntityIDs are also shown above when read for the first time

====>   Final verify status report

  Final UDF Revision range: 2.01 only

  File System info
  last valid block  : 3913727   Volume Space: 1.8662 Gbyte
  block size        : 512
  (ECC) blocking factor : 16
  nmb of sessions   : 1
  verify session    : 1
  session starts    : 0
  medium WR type    : overwritable
  medium SE type    : nonsequential
  medium CL type    : finalized

  Summed file body sizes: 337175825 bytes   (321.5559 Mbytes)

    Error count:   3    total occurrences:     3  -> search for   "error:"
  Warning count:   1    total occurrences:     1  -> search for "warning:"

  Additional notes may have been printed:     -> search for    "note:"

  Disclaimer:
-   The number of errors and warnings is an indication only.
-   There is no guarantee that the number of errors and
-   warnings as shown by the UDF verifier is correct.
```

Výpis D.4: Výsledek kontroly poškozeného média programem `fsck_udf`

```
fsck_udf: * * * File <0, 989> seems to have already-allocated blocks! * * *
fsck_udf: * * * File <0, 989> seems to have fewer blocks (2116354) than expected
    (19202) * * *
fsck_udf: * * * File <0, 989> seems to have fewer data bytes (1083573248) than
    expected (9830400) * * *
fsck_udf: * * * File <0, 989> seems to have fewer bytes (1083573248) than expected
    (9830400) * * *
1 errors total
** Checking Hierarchy
** Filesystem error: Bad extent in file
** Filesystem error: 2017-01-16-191632.webm
** Filesystem is dirty and non-repairable
LVID = {
    blkNum = 128
    numFiles = 2
    numDirs = 1
    minReadRevision = 201
    minWriteRevision = 201
    maxWriteRevision = 201
    numFreeBlks[0] = 3253614
    numFreeBlks[1] = 0
    numFreeBlks[2] = 0
    numTotalFreeBlks = 3253614
    numPart = 1
    nextUniqueId = 0
}
```

E OBSAH PŘILOŽENÉHO DVD

- `/` ..... kořenový adresář archivu
  - `udftools` ..... Adresář balíčku `udftools` obsahující nástroj `udffsck`
    - `AUTHORS`
    - `autogen.sh` ..... Skript pro kompilaci balíku
    - `cdrwtool` ..... Adresář se zdrojovými kódy nástroje `cdrwtool`
    - `configure.ac`
    - `COPYING`
    - `doc` ..... Adresář s dokumentacemi k jednotlivým nástrojům
      - `udffsck.8` ..... Manuálová stránka k nástroj `udffsck`
      - `fsck.udf.8` ..... Alias na udffsck.8
      - `html` . Dokumentace v html podobě vygenerovaná nástrojem Doxygen
        - `index.html` ..... Vstupní bod dokumentace `.3 Doxyfile`
    - `ChangeLog`
    - `include` ..... Adresář se sdílenými hlavičkovými soubory
      - `bswap.h`
      - `ecma_167.h` ..... Soubor se strukturami podle ECMA-167
      - `libudffs.h` .. Soubor s definicemi funkcí ze sdílené knihovny `libtool`
      - `osta_udf.h` .. Soubor se strukturami podle OSTA UDF dokumentace
      - `udf_endian.h`
      - `udf_lib.h`
    - `INSTALL`
    - `libudffs` ..... Adresář se zdrojovými kódy sdílené knihovy `libudffs`
    - `Makefile.am`
    - `mkudffs` ..... Adresář se zdrojovými kódy nástroje `mkudffs`
    - `NEWS`
    - `pktsetup`
    - `README`
    - `README.md`
    - `TODO`
    - `udffsck` ..... Adresář se zdrojovými kódy nástroje `udffsck`
      - `main.c` ..... Hlavní soubor programu.
      - `Makefile.am`
      - `options.c` ..... Parser vstupních parametrů a nápověda.
      - `options.h`
      - `test.c` ..... Zdrojový soubor pro testy.
      - `udffsck.c` ..... V tomto souboru je jádro celého programu.
      - `udffsck.h`
      - `utils.c` ..... Podpůrné funkce (například tisk tagů)
      - `utils.h`
    - `wrudf` ..... Adresář se zdrojovými kódy nástroje `wrudf`
  - `udf-samples.tar.xz` ..... Komprimovaný archiv s testovacími daty
  - `udf-samples-extra.tar.xz` ..... Komprimovaný archiv s rozšířenými testovacími daty

- `udffsck-outputs` ..... Výstupy z nástroje `udffsck`. K prohlížení doporučuji nástroj `less`
- `vladyka.pdf` ..... Tato práce v elektronické podobě
- `demo.avi` ..... Záznam demonstrující činnost kontroly a opravy média